PENGEMBANGAN PERANGKAT MIDDLEWARE WEB OF THINGS (WOT) BERBASIS ARSITEKTUR PUBLISH SUBSCRIBE MENGGUNAKAN PROTOKOL REST HTTP

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Donny Kurniawan
NIM: 135150207111098

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
2018

PENGESAHAN

PENGEMBANGAN PERANGKAT MIDDLEWARE WEB OF THINGS (WOT) BERBASIS ARSITEKTUR PUBLISH SUBSCRIBE MENGGUNAKAN PROTOKOL REST HTTP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Donny Kurniawan
NIM: 135150207111098

Skripsi ini telah diuji dan dinyatakan lulus pada
06 Juni 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Adhitya Bhawiyuga, S.Kom, M.S	Reza Andria Siregar, S.T., M.Kom.
NIK: 201405 890720 1 001	NIP: 19790621 200604 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 22 April 2018

Donny Kurniawan

NIM: 135150207111098

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena berkat rahmat serta bimbingan-Nya, penulis dapat menyelesaikan penulisan skripsi dengan baik. Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan untuk memperoleh gelar Sarjana Komputer pada Proram Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya. Judul yang penulis ajukan adalah Pengembangan Perangkat Middleware Web Of Things (WOT) Berbasis Arsitektur Publish Subscribe Menggunakan Protokol HTTP.

Dalam penyusunan dan penulisan skripsi ini tidak terlepas dari bantuan, bimbingan serta dukungan dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis dengan senang hati menyampaikan terima kasih kepada:

1. Bapak Sukardi dan Sustiana selaku orang tua penulis yang selalu memberikan dukungan dan doa untuk penulis agar dapat menyelesaikan skripsi ini.
2. Bapak Adhitya Bhawiyuga, S.Kom, M.S selaku dosen pembimbing I yang telah dengan sabar, tulus, ikhlas dan tekun dalam meluangkan waktu, tenaga dan pikiran dalam memberikan bimbingan, motivasi, arahan dan saran terhadap penulis selama menyusun skripsi.
3. Bapak Reza Andria Siregar, S.T, M.Kom dan pembimbing II yang telah dengan sabar, tulus, ikhlas dan tekun dalam meluangkan waktu, tenaga dan pikiran dalam memberikan bimbingan, motivasi, arahan dan saran terhadap penulis selama menyusun skripsi.
4. Seluruh Dosen Program Studi Informatika Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis.
5. Teman–teman angkatan 2013 Informatika, geng kopma, terima kasih atas segala bantuan selama menempuh studi di Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
6. Annisa Septiana Sani, S.Kom yang selalu memberikan dukungan serta motivasi untuk penulis.
7. Seluruh pihak yang telah membantu kelancaran penulisan skripsi yang tidak dapat penulis sebutkan satu persatu.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan. Penulis mengharapkan adanya saran dan kritik membangun dari para pembaca demi kesempurnaan skripsi ini. Semoga skripsi ini dapat memberikan manfaat bagi semua dan berguna bagi pengembangan ilmu pengetahuan.

Malang, 22 April 2018

Penulis

doniiawan@gmail.com

ABSTRAK

Internet of Things (*IoT*) merujuk kepada semua benda atau perangkat fisik yang terhubung dan bertukar data melalui Internet. Dengan penerapan *IoT* dalam berbagai bidang, tentunya tidak lepas dari tantangan yang semakin besar pula. Salah satu masalah utama dalam penerapan IoT antara lain perangkat dengan protokol yang beragam dan aksesibilitas dari berbagai perangkat. Salah satu solusi yang dapat diberikan untuk menyelesaikan masalah interoperabilitas adalah dengan menggunakan web sebagai platform integrasi universal. Akan tetapi penerapan WoT dalam pengiriman perintah terhadap perangkat IoT menjadi tidak efisien dikarenakan perangkat IoT melakukan *request* berkala sebagai mekanisme pengecekan perintah. Oleh karena itu diperlukan penggantian arsitektur yang lebih efisien dalam mengatasi pengiriman perintah *middleware* terhadap *node sensor*. Untuk mengatasi masalah tersebut diperlukan penggantikan HTTP RESTful yang berarsitektur *request response* menjadi menggunakan arsitektur *publish subscribe*. Berdasarkan penjelasan sebelumnya diusulkan sebuah pembangunan *middleware* dengan arsitektur *publish subscribe* sebagai komunikasi perangkat *node sensor* terhadap *middleware*. Dalam penggantian arsitektur tersebut diharapkan dapat mengurangi beban dari perangkat *IoT* sehingga tidak perlu melakukan *request* secara berkala. Hasil pengujian performa sistem menunjukan, sistem yang dibangun mampu menangani hingga 150 pengguna dengan tingkat *error rate* 0%.

Kata kunci: *IoT, WoT, publish subscribe, interoperabilitas, RESTful, HTTP*.

ABSTRACT

Internet of Things (IoT) refers to all physical objects or devices that connect and exchange data over the Internet. With the application of IoT in various fields, of course cannot be separated from the increasing challenge also. One of the major problems in IOT deployment includes devices with diverse protocols and accessibility of various devices. One solution that can be given to solving interoperability problems is to use the web as a universal integration platform. However, the application of WoT in sending commands to IoT devices becomes inefficient because the IoT device makes periodic requests as a command check mechanism. Therefore, it is necessary to replace the more efficient architecture in overcoming the delivery of middleware commands to the sensor nodes. To overcome the problem is required replacement HTTP RESTful request response architecture to use publish subscribe architecture. Based on the previous explanation, it is proposed a middleware development with publish subscribe architecture as communication of sensor node device to middleware. In replacement of the architecture is expected to reduce the load from the IoT device, so no need to make requests on a regular basis. System performance test results show, the system capable of handling up to 150 users with a 0% error rate.

Keywords: IoT, WoT, publish subscribe, interoperability, RESTful, HTTP.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Internet of Things (*IoT*) merujuk kepada semua benda atau perangkat fisik yang terhubung dan bertukar data melalui Internet (Patel & Patel, 2016). *IoT* memiliki komponen yang berperan penting dalam memberdayakan fungsionalitasnya antara lain *identification, sensing, communication, computation, services* dan *semantics* (Al-Fuqaha, et al., 2015). Komponen-komponen tersebut membuat perangkat fisik dapat diidentifikasi, saling bertukar data dan memiliki kecerdasan dalam membuat sebuah keputusan berdasarkan dari informasi yang didapatkan. Dengan penerapan *IoT* dalam berbagai bidang, tentunya tidak lepas dari tantangan yang semakin besar pula. Tantangan yang dihadapi *IoT* meliputi penggunaan sumber daya yang terbatas, skalabilitas, heterogenitas objek, kapasitas penyimpanan yang terbatas (Farahzadi, et al., 2017). Xiao, et al. (2014) menyatakan terdapat masalah utama dalam penerapan IoT antara lain perangkat dengan protokol yang beragam dan aksesibilitas dari berbagai perangkat. Oleh karena itu, dibutuhkan sebuah sistem yang dapat mengatasi interoperabilitas antar berbagai macam perangkat *IoT*.

Salah satu solusi yang dapat diberikan untuk menyelesaikan masalah interoperabilitas adalah dengan menggunakan web sebagai platform integrasi universal. Hal tersebut menjadikan sebuah istilah baru yang bernama *Web of Things* (*WoT*). *Web of Things* adalah sebuah mekanisme dimana perangkat *IoT* dapat diakses langsung sebagai *web resources* (Mainetti, et al., 2015). Salah satu keuntungan menggunakan standar web adalah bahwa perangkat pada akhirnya dapat berkomunikasi dengan mekanisme yang sama terhadap sumber daya lainnya di *Internet,* sehingga hal tersebut memudahkan integrasi objek fisik terhadap halaman web (Guinard & Trifa, 2009). Dengan adanya WoT diharapkan pengembang aplikasi berbasis IoT tidak perlu repot dalam mengatasi masalah interoperabilitas dari berbagai macam perangkat.

REST adalah sebuah representasi dalam mendapatkan sebuah *resource* melalui protokol HTTP (Khare & Taylor, 2004). REST merupakan salah satu webservice yang umum digunakan dalam penerapan *WoT*. REST umum digunakan dalam WoT karena memiliki kelebihan antara lain, REST memiliki *self descriptive messages*, *stateless operation* dan mendukung berbagai representasi data (Paganelli, et al., 2016). Adapun pendekatan *WoT* terbagi menjadi dua bagian yaitu *direct integration* dan *indirect integration*. *Direct intergration* adalah penerapan WoT dengan cara menanamkan REST server kedalam perangkat IoT, sedangkan *indirect integration* dengan cara memberikan sebuah perangkat *middleware* sebagai perantara komunikasi antara perangkat IoT dengan web (Zeng, et al., 2011).

Pada penelitian sebelumnya yang dilakukan oleh Gao, et al. (2011), peneliti merancang sebuah *middleware WoT* dengan menggunakan protokol HTTP RESTful sebagai komunikasi *middleware* terhadap perangkat node sensor.

Penggunaan HTTP RESTful yang berbasis arsitektur *request response* tersebut memiliki kelemahan dalam mengirimkan sebuah perintah kepada node sensor. Kelemahan yang dimaksud adalah node sensor akan melakukan sebuah *request* berkala sebagai mekanisme pengecekan perintah. Tentunya hal ini membuat beban kinerja node sensor menjadi lebih berat. Oleh karena itu diperlukan penggantian arsitektur yang lebih efisien dalam mengatasi pengiriman perintah *middleware* terhadap node sensor.

Untuk mengatasi masalah tersebut peneliti menggunakan arsitektur *publish subscribe* sebagai pengganti arsitektur *request response*. Prinsip dari model komunikasi *publish subscribe* yaitu memiliki beberapa komponen utama yaitu *publisher*, *subscriber* dan *broker* (Hunkeler, et al., 2015). *Publisher* bertugas sebagai pengirim informasi, *subscriber* sebagai penerima informasi, dan *broker* sebagai penerus informasi yang dikirimkan dari publisher ke subscriber. Dalam pengiriman dan penerimaan data, ditambahkan sebuah mekanisme dalam mendapatkan informasi sesuai dengan keminatan atau yang biasa disebut dengan topik. Jika terdapat kecocokan antara topik *publisher* dan topik *subscriber* maka *broker* akan meneruskan pengiriman data terhadap *subscriber*. Penggantian arsitektur ini dapat mempermudah pengiriman sebuah perintah terhadap *node sensor* yang sebelumnya melakukan *request* secara berkala, diganti dengan menggunakan sebuah komponen *broker* yang berfungsi secara otomatis meneruskan perintah kepada node sensor.

Dari permasalahan yang sudah dijelaskan sebelumnya maka peneliti membangun *middleware* dengan arsitektur *publish subscribe* sebagai komunikasi perangkat *node sensor* terhadap *middleware*. Dalam penggantian arsitektur tersebut diharapkan dapat mengurangi beban dari perangkat *IoT* sehingga tidak perlu melakukan *request* secara berkala. Adapun tujuan lainnya dari pembangunan *middleware* ini adalah tidak menghilangkan konsep *WoT* dalam *middleware* yang akan dibuat, sehingga pengguna dapat mengakses data sensor serta memberikan perintah terhadap perangkat *IoT* dengan menggunakan antarmuka RESTful. Terdapat 4 komponen utama dalam *middleware* yang akan dibuat meliputi: komponen REST server, komponen *database*, komponen *subscriber middleware* dan komponen *node sensor*. Komponen REST server merupakan komponen yang menangani komunikasi user melalui protokol HTTP. Komponen *database* merupakan komponen yang menangani dalam hal penyimpanan data, seperti data sensor dan *list* node sensor yang terdaftar. Komponen *subscriber middleware* adalah komponen yang bertugas mendapatkan dan meneruskan data sensor ke dalam database dengan menggunakan arsitektur *publish subscribe*. Komponen node sensor adalah perangkat IoT yang berkomunikasi menggunakan arsitektur *publish subscribe* terhadap *middleware*.

1.2 Rumusan Masalah

Adapun rumusan masalah yang mendasari penulis dalam melakukan penelitian adalah:

1. Bagaimana alur data dari pengguna sampai ke perangkat *node sensor*?
2. Bagaimana alur data dari node sensor sampai ke pengguna?
3. Bagaimana mekanisme translasi perintah dari REST ke MQTT di dalam *middleware?*
4. Bagaimana performansi dan kinerja fungsional REST *middleware* berbasis *publish subscribe?*

1.3 Tujuan

Adapun tujuan dari penelitian yang akan dilakukan antara lain:

1. Menggantikan arsitektur komunikasi *request response* dengan arsitektur *publish subscribe* MQTT pada *node sensor*.
2. Menerapkan arsitektur *publish subscribe* sebagai pengganti arsitektur *request response* terhadap *node sensor*.
3. Mengetahui kinerja REST *middleware* berbasis arsitektur publish/*subscribe* dari segi fungsionalitas dan performa

1.4 Manfaat

Adapun manfaat yang dicapai dalam pengerjaan penelitian ini yaitu:

1. Bagi Penelitian ini diharapkan dapat menjadi salah satu acuan referensi dalam membangun sistem REST *middleware* terhadap interoperabilitas berbagai macam perangkat *IoT* (*node sensor*).
2. Penelitian ini menjadi wawasan mengenai proses translasi 2 protokol pada *IoT middleware* yang dapat menjadi acuan untuk mengembangkan penelitian yang memiliki keterkaitan dengan skripsi ini.

1.5 Batasan Masalah

Batasan masalah dalam penelitian ini terdapat beberapa batasan permasalahan, adapun sebagai berikut:

1. Protokol komunikasi data yang digunakan pada penelitian ini adalah HTTP RESTful dan MQTT.
2. *Platform IoT node sensor* yang digunakan adalah Raspberry Pi, *Node*MCU dan Wemos D1.
3. Format data yang digunakan dalam merepresentasikan data adalah JSON.
4. VirtualBox berperan sebagai Server *Middleware*.
5. Data yang dikirimkan berupa data *sensor* pada masing-masing *node* (*Node*MCU, Wemos D1 dan Raspberry PI).
6. Dalam penelitian ini mengabaikan sisi keamanan pada protokol tersebut.

1.6 Sistematika Pembahasan

Dalam penyusunan skripsi ini, struktur penulisan yang digunakan beserta penjelasannya adalah sebagai berikut:

BAB 1 : Pendahuluan

Menguraikan masalah yang diangkat dari hal yang umum pada IoT hingga masalah spesifik menjurus pada masalah utama latar belakang pengerjaan skripsi ini dan sedikiti penjelasan dari penelitian yang akan dilakukan. Bab ini terdiri dari: latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan.

BAB 2 : Landasan Kepustakaan

Membahas kajian pustaka dan penelitian sebelumnya yang berhubungan dengan latar belakang serta metode yang digunakan pada skripsi ini. Kajian pustaka ini berasal dari referensi-referensi berkaitan dan mendukung dalam penelitian ini.

BAB 3 : Metodologi

Menguraikan mengenai metode serta langkah kerja penelitian yang terdiri atas studi literatur, analisis kebutuhan, perancangan sistem, implementasi, pengujian serta pengambilan kesimpulan dari penelitian yang telah dilakukan.

BAB 4 : Analisis Kebutuhan dan Perancangan

Membahas tentang kebutuhan sistem, seperti kebutuhan fungsional, non-fungsional, perangkat keras, perangkat lunak dan menjelaskan rancangan sistem.

BAB 5 : Implementasi

Menjelaskan proses implementasi sesuai dengan analisis kebutuhan dan perancangan sistem.

BAB 6 : Pengujian

Pengujian dilakukan jika perancangan dan implementasi telah selesai dilakukan. Pengujian bertujuan untuk melakukan uji coba sistem, serta mengetahui kinerja sistem.

BAB 7 : Penutup

Memuat kesimpulan dari pembuatan dan pengujian sistem, dan saran untuk pengembangan atau penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini berisi teori-teori yang dikumpulkan untuk menjadi acuan pada studi literatur dalam mendukung penelitian "Pengembangan REST *Middleware* Berbasis Arsitektur *Publish-Subscribe*". Adapun pencarian dan pengumpulan studi literatur merupakan tahap awal dalam melakukan penelitian yang akan dibangun.

2.1 Kajian Pustaka

Pada penelitian yang dilakukan oleh Mainetti, et al. (2015) berjudul "***A Software Architecture Enabling the Web of Things***" menyatakan bahwa solusi yang dapat diberikan untuk menyelesaikan masalah interoperabilitas antar berbagai macam perangkat *IoT* adalah dengan menggunakan web sebagai platform integrasi universal. Pada penelitian selanjutnya yang dilakukan oleh Gao, et al. (2011) berjudul "***RESTful Web of Things API in Sharing Sensor Data***" peneliti merancang sebuah middleware *WoT* dengan menggunakan protokol HTTP RESTful pada komunikasi *middleware* terhadap *node sensor*. Penggunaan HTTP RESTful yang berbasis arsitektur *request response* tersebut memiliki kelemahan dalam mengirimkan sebuah perintah kepada *node sensor*. Kelemahan yang dimaksud adalah *node sensor* akan melakukan sebuah *request* berkala sebagai mekanisme pengecekan perintah. Tentunya hal ini membuat beban kinerja node sensor menjadi lebih berat.

2.2 Internet of Things

Internet of Things (*IoT*) merujuk kepada semua benda atau perangkat fisik yang terhubung dan bertukar data melalui Internet (Patel & Patel, 2016). *IoT* memungkinkan sebuah perangkat fisik dapat melihat, mendengar, berpikir, mengambil keputusan dengan mengharuskan mereka berbagi informasi dengan perangkat fisik lainnya. *IoT* merubah benda tersebut yang awalnya hanya benda mati menjadi sebuah objek yang "pintar". *IoT* memiliki komponen yang berperan penting dalam memberdayakan fungsionalitasnya antara lain *identification, sensing, communication, computation, services* dan *semantics* (Al-Fuqaha, et al., 2015). Komponen-komponen tersebut membuat perangkat fisik dapat diidentifikasi, saling bertukar data dan memiliki kecerdasan dalam membuat sebuah keputusan berdasarkan dari informasi yang didapatkan.

Identification adalah sebuah proses untuk menyediakan identitas yang jelas bagi setiap objek yang terdapat pada sebuah jaringan. *Sensing* berarti perangkat *IoT* dapat mengumpulkan data dari objek yang terhubung dan menyimpan data tersebut ke *cloud* atau *database*. *Communication* berarti perangkat *IoT* menghubungkan berbagai macam objek dan secara bersama membentuk sebuah layanan. *Computation* berarti perangkat IoT memiliki *processing unit* atau perangkat lunak yang merepresentasikan "otak" dan kemampuan komputasi perangkat *IoT*. *Service* berarti perangkat *IoT* mengambil keputusan pada data yang telah diperoleh serta memberikan layanan akses terhadap data tersebut.

Semantics berarti perangkat *IoT* mengambil pengetahuan secara tepat dari berbagai objek dalam menyediakan layanan yang dibutuhkan (Al-Fuqaha, et al., 2015).

2.3 Web of Things

Web of things mengacu kepada perubahan perkembangan IoT yang sebelumnya Internet of Things yaitu benda-benda yang terhubung ke Internet, menjadi Web of Things. *Web of Things* adalah sebuah mekanisme dimana perangkat *IoT* dapat diakses langsung sebagai *web resources* (Mainetti, et al., 2015). Tujuan utama dalam konsep *WoT* ini adalah menambah interoperability terhadap berbagai macam perangkat *IoT*, sehingga memudahkan dalam melakukan akses data, melakukan *programming* dan lain-lain. Salah satu keuntungan menggunakan standar web adalah bahwa perangkat pada akhirnya dapat berkomunikasi dengan mekanisme yang sama terhadap sumber daya lainnya di Internet, sehingga hal tersebut memudahkan integrasi objek fisik terhadap halaman web (Guinard & Trifa, 2009). Diharapkan dengan adanya *WoT,* pengembang aplikasi hanya perlu berfokus dalam pengembangan aplikasinya tanpa memikirkan soal protokol komunikasi yang akan digunakan. Adapun pendekatan *WoT* terbagi menjadi dua bagian yaitu *direct integration* dan *indirect integration*. *Direct intergration* adalah penerapan WoT dengan cara menanamkan REST server kedalam perangkat IoT, sedangkan *indirect integration* dengan cara memberikan sebuah perangkat *middleware* sebagai perantara komunikasi antara perangkat IoT dengan web (Zeng, et al., 2011).

2.4 Bahasa Pemrograman

Dalam membangun suatu aplikasi atau perangkat lunak dibutuhkan suatu bahasa yang dapat menengahi antara *programmer* dan komputer. Bahasa yang biasa digunakan oleh *programmer* disebut bahasa pemrograman, yaitu bahasa yang dipakai dalam membangun suatu perangkat lunak yang menggunakan alur kerja berupa prinsip algoritma yang sudah dipelajari sebelumnya oleh *programmer.* Terdapat banyak tipe bahasa pemrograman yang sudah ada namun setiap bahasa pemrograman memiliki kode yang berbeda-beda pula. Kode tersebutlah yang menjadi dasar dalam menjalankan berbagai macam proses berdasarkan tipe data yang dimasukkan oleh *programmer.*

2.4.1 Python

Python adalah bahasa pemrograman interpretatif multiguna. Tidak seperti bahasa lain yang susah untuk dibaca dan dipahami, Python lebih menekankan pada keterbacaan kode agar lebih mudah untuk memahami sintaks. Hal ini membuat Python sangat mudah dipelajari baik untuk pemula maupun untuk yang sudah menguasai bahasa pemrograman lain (Belajarpython, 2017). Adapun kelebihan Python adalah dapat meningkatkan produktivias dan menghemat waktu bagi para *programmer*. Untuk memperoleh hasil program yang sama, *source code* Python juga jauh lebih sedikit dibandingkan dengan kode yang ditulis

menggunakan bahasa-bahasa pemrograman lain seperti C, C++, C# maupun Java (Nurohman, 2016).

2.4.2 C Language

Bahasa Pemrograman C adalah sebuah bahasa pemrograman komputer yang bisa digunakan untuk membuat berbagai aplikasi (*general-purpose programming language*), mulai dari sistem operasi (seperti Windows atau Linux), *antivirus*, *software* pengolah gambar (*image processing*), hingga *compiler* untuk bahasa pemrograman, di mana C banyak digunakan untuk membuat bahasa pemrograman lain, salah satunya adalah PHP. Meskipun termasuk *general-purpose programming language*, yakni bahasa pemrograman yang bisa membuat berbagai aplikasi, bahasa pemrograman C paling cocok merancang aplikasi yang berhubungan langsung dengan sistem operasi dan hardware. Ini tidak terlepas dari tujuan awal bahasa C dikembangkan (Andre, 2017).

2.4.3 Lua

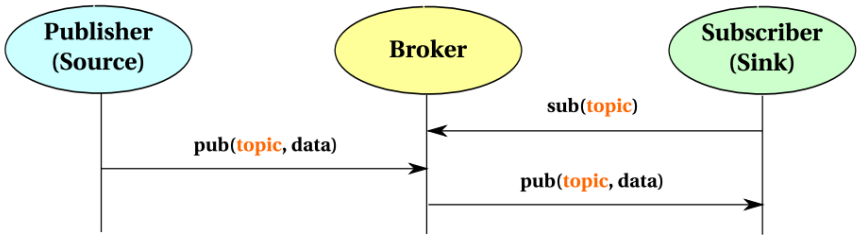
Lua adalah bahasa *scripting* yang *powerful*, efisien, ringan dan dapat disematkan dalam *embedded* sistem. Bahasa ini mendukung beberapa model pemrograman antara lain pemrograman prosedural, pemrograman berorientasi obyek, pemrograman fungsional, pemrograman berbasis data, dan deskripsi data. Lua menggabungkan sintaks prosedural sederhana dengan konstruksi deskripsi data yang kuat berdasarkan *array* asosiatif. Lua dijalankan dengan menafsirkan *bytecode* dengan mesin virtual berbasis *register* dan memiliki manajemen memori dengan pengumpul sampah otomatis. Hal ini membuatnya menjadi Bahasa pemrograman yang ideal untuk konfigurasi, *scripting*, dan *prototyping* yang cepat (Lua, 2017).

2.5 Arsitektur Komunikasi Publish/Subscribe

Prinsip dari model komunikasi *publish subcribe* yaitu beberapa komponen yang menginginkan informasi berdasarkan "keminatan" dengan cara mendaftarkan keinginan mereka atau biasa disebut dengan "topik". Proses mendaftarkan topik ini disebut *subscribe*. Ada tiga macam komponen yang utama dalam arsitektur komunikasi ini ialah *subscriber*, *publisher* dan *broker*. *Subscriber* adalah komponen yang menerima sebuah informasi berdasarkan topik yang didaftarkan, *publisher* adalah komponen yang mengirimkan informasi berdasarkan topik terkait yang diinginkan *subscriber*, sedangkan *broker* adalah komponen yang bertugas untuk memastikan paket dari *publisher* menuju ke *subscriber* dapat tersampaikan (Hunkeler, et al., 2015)

Terdapat tiga model *pub/sub: topic-based, context based, type based*. Pada *topic* based daftar dari beberapa topik sudah diketahui contoh pada saat mendesain sebuah aplikasi. Proses *subscribe* dan *publish* dilakukan pada beberapa topik tertentu saja. *Type based* tidak umum digunakan pada system saat ini. *Context based* adalah yang paling umum digunakan, *subscriber* mendeskripsikan informasi yang diinginkan namun dalam menerapkan konten

tujuan *context-based publish/subscribe* mengharuskan data ditambahkan dengan *meta-data* pada *overhead* (Hunkeler, et al., 2015).



Gambar 2.1 Mekanisme Publish Subscribe Model Topic Based

Sumber : (Hunkeler, et al., 2015)

Gambar 2.1 menggambarkan model komunikasi *publish/subcribe topic-based*. *Subscriber* akan mengirimkan perintah *sub (topic)* untuk menisyaratkan kepada *broker* tentang topik apa saja yang diinginkan oleh *subscriber,* selanjutnya publisher mengirimkan perintah *pub (topic, data)* yang didalamnya berisi topik dan data yang akan dipublikasikan ke *broker*. Jika terdapat kesamaan terhadap topik yang diminta oleh *subscriber* dan topik yang dikirimkan oleh *publisher* maka *broker* akan mengirimkan perintah *pub (topic, data)* menuju *subscriber*. Sebuah perintah *pub* dapat didistribusikan ke beberapa *subscriber* sekaligus apabila terdapat kesamaan terhadap topik yang diinginkan.

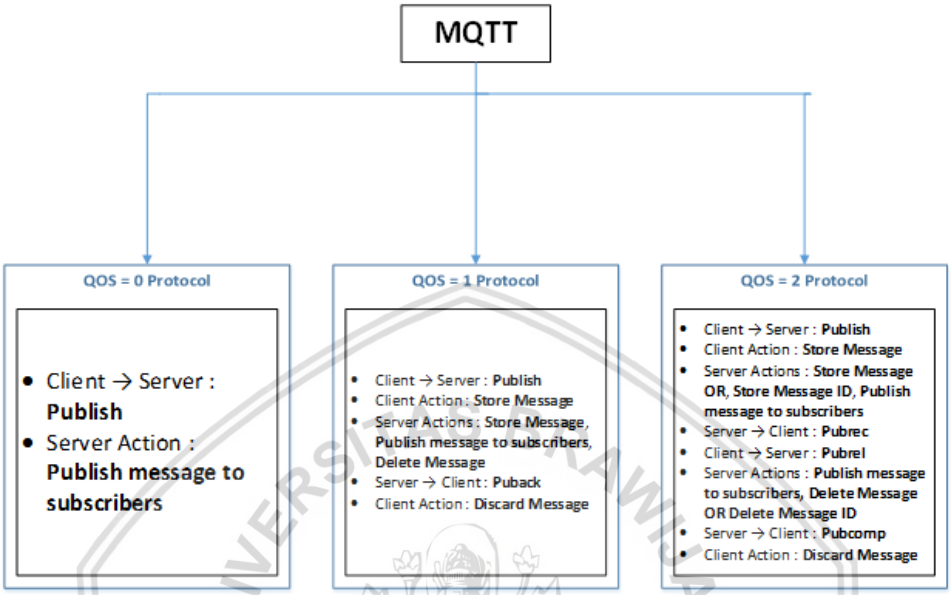
2.5.1 MQTT

MQTT adalah singkatan dari *Message Queuing Telemetry Transport*. MQTT diciptakan pada tahun 1999 oleh Dr Andy Stanford-Clark dan Arlen Nipper. MQTT adalah protokol komunikasi client server *publish/subscribe* berbasis topik yang ringan, *open source* dan mudah diterapkan (MQTT, 2014). Karakteristik ini membuatnya ideal untuk digunakan dalam banyak situasi, termasuk lingkungan terbatas seperti untuk komunikasi dalam *Machine to Machine (M2M)* dan *Internet of Things (IoT)*. MQTT sangat ideal untuk perangkat yang terbatas maupun aplikasi *mobile* di era M2M/*IoT* di mana *bandwidth* dan daya baterai menjadi pertimbangan utama (Al-Fuqaha, et al., 2015).

MQTT merupakan protokol yang menggunakan sebuah karakter *string* dalam menentukan hirarkial dari sebuah topik. Sebagai contoh sebuah sensor suhu yang berlokasi di lantai "F2", ruang "R248" maka sensor tersebut sebaiknya menggunakan topik "wsn/sensor/F2/R248/temperature". Karakter "/" digunakan untuk memisahkan setiap bagian dari topik. Selain itu terdapat beberapa karakter *wildcard* yang menandakan "seluruh bagian" dari sebuah topik. Contoh dalam topik "wsn/sensor/F2/+/temperature" yang memiliki arti bahwa topik yang diinginkan adalah seluruh data sensor suhu yang berada lantai "F2".

MQTT juga memiliki tiga macam QoS dalam upaya mengandalkan pengiriman datanya. MQTT menawarkan tiga jenis mode yang disebut Quality of Service (QoS). Pada mode QoS 0 pengirim paket hanya mengirimkan satu kali tanpa

memerlukan pesan konfirmasi atau ACK. Pada mode QoS 1 memastikan bahwa pesan disampaikan setidaknya satu kali dengan membutuhkan ACK, akan tetapi pada QoS 1 memungkinkan terjadi duplikat data pada penerima. Mode QoS 2 menjamin bahwa pesan disampaikan dengan tepat sekali tanpa terjadi duplikat data pada penerima (Amaran, et al., 2015).



Gambar 2.2 MQTT QoS

(Sumber : (Aziz, 2015))

2.6 REST

REST adalah singkatan dari *REpresentational State Transfer*. REST adalah sebuah representasi dalam mendapatkan sebuah *resource* melalui protokol HTTP (Khare & Taylor, 2004). Konsep REST pertama kali diperkenalkan oleh Roy Fielding pada thesis yang dilakukannya ditahun 2000. Cara kerjanya, REST *server* menyediakan jalur untuk akses *resource*, sedangkan REST *client* melakukan akses *resource* dan kemudian menampilkannya. REST memiliki beberapa komponen antara lain *resource*, URI, representasi, dan jenis *method* HTTP *request*.

REST memodelkan transfer data antar komponen berbasis *resource*, dimana setiap *resource* dipetakan kedalam URI (*Uniform Resource Identifier*) yang unik. Dalam representasi data sebuah *resource* dapat berupa format CSV, XML maupun objek JSON. Dalam mengakses sebuah *resource*, REST server menyediakan beberapa *method* yang diperlukan dalam mengubah sebuah keadaan *resource* tersebut antara lain GET, POST, PUT dan DELETE (Sandoval, 2009). Adapun dari beberapa method tersebut akan dipetakan dalam Tabel 2.1:

Tabel 2.1 Method RESTful

Method	Aksi
GET	Menampilkan sebuah *resource*

POST	Menambahkan sebuah *resource*
PUT	Mengubah sebuah *resource*
DELETE	Menghapus sebuah *resource*

Sumber : Sandoval (2009)

2.7 Flask

Flask adalah mikro *web-framework* ditulis dalam bahasa pemrograman Python (Flask, 2017). Yang dimaksud "mikro" adalah tidak berarti bahwa seluruh aplikasi web diharuskan masuk ke dalam satu file Python. Kata "micro" di *microframework* berarti Flask berfokus kepada kesederhanaan *framework* tersebut tetapi *extensible. Mikro-framework* biasanya berarti *framework* dengan sedikit atau tanpa *dependencies* terhadap *library external*. Flask memiliki dua komponen utama yaitu Werzeug dan Jinja2. Dalam hal routing, debugging dan Web Server Gateway Interface (WSGI) ditangani oleh Werzeug sedangkan dalam hal tampilan atau *template* ditangani oleh Jinja2 (Grinberg, 2014). Beberapa kelebihan framework flask adalah ringan, dan dengan mudahnya menambahkan *library* eksternal sesuai dengan kebutuhan developer. Dengan flask, pengembang aplikasi dapat memilih *library* maupun membuat *library* yang dibutuhkan.

2.8 Redis

Redis adalah sebuah sistem penyimpanan struktur data *key-value* yang berbasis NonSQL. Redis menggunakan ram sebagai tempat penyimpanan datanya, oleh karena itu redis dapat menghasilkan performa tinggi (Nelson, 2016). Redis banyak digunakan dalam beberapa perusahaan startup seperti Twitter dan Uber. Redis memiliki beberapa struktur data antara lain *strings, lists, hashes, sets, and sorted sets*. Pada dasarnya redis diciptakan tidak untuk menggantikan sistem basis data yang sudah ada, akan tetapi redis hanya ditujukan khusus dalam operasi temporary dan penyimpanan data dinamis (Macedo & Oliveira, 2011). Cara kerja Redis sama seperti *cache*, setiap *key* diisi oleh *value*. Redis dapat juga menjalankan operasi *atomic* seperti menambahkan *string*, membulatkan nilai pada *hash,* menambahkan sebuah elemen pada *list* dan sebagainya (Redis, 2017).

2.9 MongoDB

MongoDB adalah sebuah sistem basis data berorientasi dokumen yang *powerful*, fleksibel dan *scalable*. Sistem basis data ini menggantikan konsep "baris" menjadi sebuah "dokumen". Tidak seperti sistem basis data berorientasi "baris" yang menggunakan skema tetap dan didefinisikan diawal, sistem basis data dokumen tidak memiliki skema yang tetap. Sehingga hal tersebut menjadikan kunci dan nilai dari dokumen dapat berbeda ukuran maupun jenis. Dengan tidak adanya skema pada MongoDB diharapkan dapat memudahkan baik dalam proses penambahan maupun pengurangan data. Selain itu MongoDB juga

memiliki beberapa fitur yang dimiliki sistem basis data relational antara lain secondary indexes, range querries dan sorting (Chodorow & Dirolf, 2010).

Dokumen adalah model data bagi mongoDB yang kurang lebih seperti satu baris dalam sistem basis data relasional (Chodorow & Dirolf, 2010). Dokumen memiliki struktur data yang terdiri dari field dan nilai yang menjadikan hal tersebut identik dengan JSON. Nilai sebuah dokumen dapat berisi array, array dari dokumen maupun dokumen lain sehingga dapat mengurangi proses join seperti di sistem basis data relasional. MognoDB menyimpan datanya dalam bentuk BSON. BSON adalah sebuah representasi biner dari dokumen JSON yang memiliki lebih banyak tipe data.

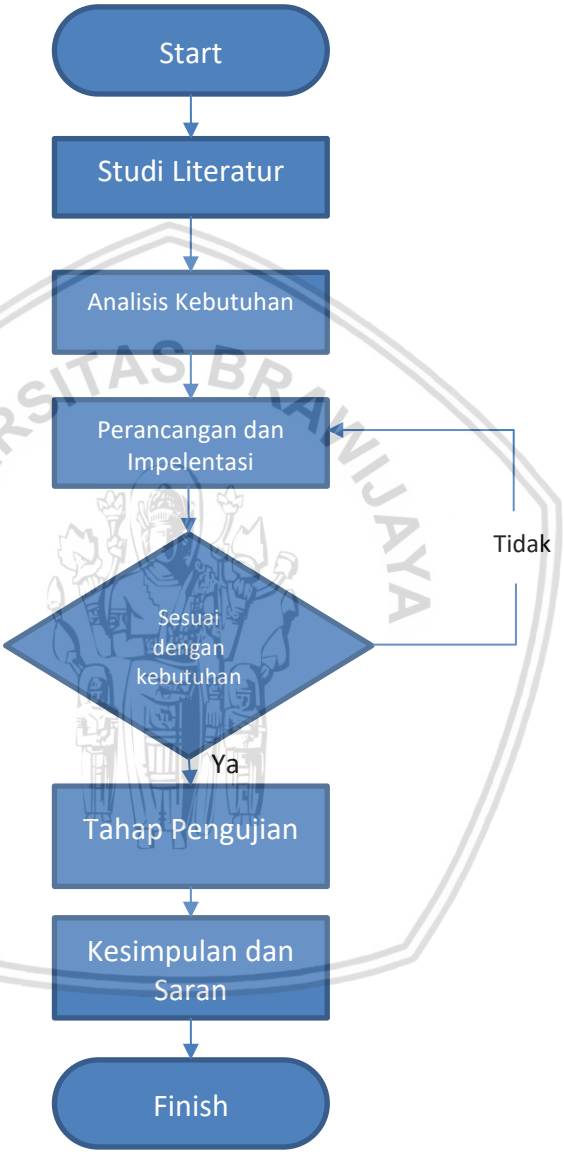
Gambar 2.3 MongoDB : Dokumen

Sumber : MongoDB(2018)

Collection adalah sekumpulan dari dokumen BSON. Collection dalam mongoDB kurang lebih sama seperti tabel dalam sistem basis data relasional (Chodorow & Dirolf, 2010). Jika tabel memiliki sebuah skema tetap, berbeda halnya dengan collection yang tidak memiliki skema tetap. Dengan tidak adanya skema membuat mongoDB menjadi sistem basis data yang memiliki fleksibilitas tinggi Hal tersebut diharapkan dapat memudahkan para pengembang aplikasi dalam melakukan penyimpanan dengan data yang berbeda.

BAB 3 METODOLOGI

Pada bab ini dijelaskan metode yang digunakan penulis dalam penelitian Pengembangan *Rest Middleware Berbasis Arsitektur Publish-Subscribe*. Adapun tahapan yang akan dilakukan adalah sebagai yang terpapar dalam diagram di bawah ini:



Gambar 3.1 Diagram Alur Metodologi Penelitian

3.1 Studi Literatur

Studi literatur dilakukan dengan merujuk jurnal dan buku yang berkaitan dengan penelitian yang dilakukan oleh penulis. Hal ini dimaksudkan agar data penelitian ini lebih valid terutama secara teoritis. Teori pendukung tersebut meliputi:

a. Internet of Things

Pada bagian ini melakukan kajian literatur tentang apa itu *Internet of Things* beserta ciri-ciri dan karakteristiknya.

b. *Web Of Things*

Pada bagian ini melakukan kajian literatur tentang apa itu *Web of Things*, permasalahan yang terdapat dalam *Web of Things* dan bagaimana cara mengatasi solusinya.

c. Python

Pada bagian ini melakukan kajian literatur tentang Bahasa pemrograman python.

d. C *Language*

Pada bagian ini melakukan kajian literatur tentang Bahasa pemrograman C.

e. Lua

Pada bagian ini melakukan kajian literatur tentang Bahasa pemrograman Lua.

f. Arsitektur *Publish Subscribe*

Pada bagian ini melakukan kajian literatur tentang arsitektur komunikasi publish subscribe dan bagaimana cara kerjanya.

g. REST

Pada bagian ini melakukan kajian literatur tentang REST API dan bagaimana cara kerjanya.

h. Flask

Pada bagian ini melakukan kajian literatur terhadap aplikasi *web* berbasis framework Flask dengan menggunakan Bahasa pemrograman Python, beserta apa saja keunggulan framework ini.

i. Redis

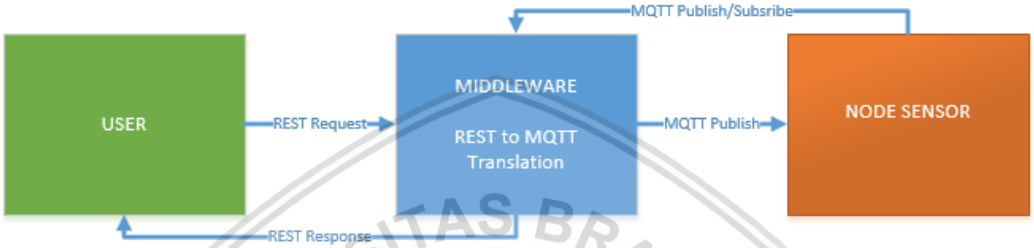
Pada bagian ini melakukan kajian literatur terhadap sistem basis data Redis, macam-macam tipe datanya dan bagaimana cara menggunakannya.

j. MongoDB

Pada bagian ini melakukan kajian literatur terhadap sistem basis data mongoDB, macam-macam tipe datanya dan bagaimana cara menggunakannya.

3.2 Analisis Kebutuhan Sistem

Analisis kebutuhan merupakan salah satu sebuah cara dalam mempersiapkan penelitian. Tujuan dari analisis kebutuhan ini adalah untuk mengetahui perangkat keras dan perangkat lunak yang dibutuhkan untuk penelitian ini. Adapun kebutuhan yang wajib dipenuhi pada penelitian ini adalah kebutuhan perangkat keras seperti laptop dan perangkat lunak seperti *virtual machine*.

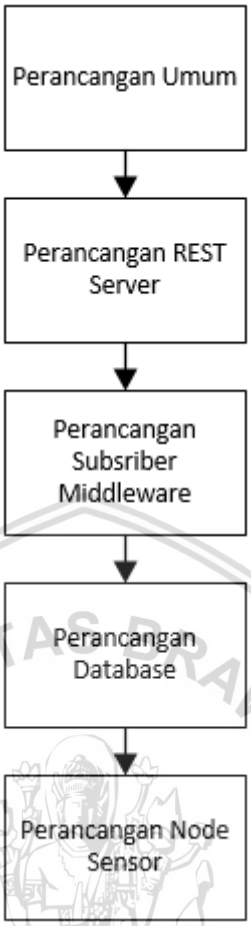


Gambar 3.2 Gambaran Umum Sistem

Terdapat tiga entitas utama dalam sistem yang akan dibangun yaitu *user*, *middleware*, dan *node sensor*. User adalah pengguna yang mengakses REST server melalui web browser. Middleware mencakup REST server, MQTT broker, *database* Redis, *database* MongoDB dan *subscriber middleware* yang berjalan pada *virtual machine* dan telah terhubung pada jaringan. Sedangkan *node sensor* berupa tiga perangkat *IoT* yaitu Raspberry Pi, NodeMCU, dan Wemos D1. *User* dapat mengirim perintah ke *middleware* dengan cara mengirimkan HTTP *request* ke *middleware,* ketika *middleware* menerima sebuah HTTP *request middleware* melakukan *publish dengan* menggunakan protokol komunikasi MQTT kepada *broker* MQTT*,* kemudian *broker* MQTT meneruskan *publish* data kepada *node sensor* yang terhubung. Pada intinya *middleware* melakukan translasi dari protokol HTTP kedalam MQTT baik dalam mengirim perintah maupun menerima data terhadap *node sensor*.

3.3 Perancangan Sistem

Perancangan dilakukan setelah semua kebutuhan sudah terpenuhi melalui tahap rekayasa kebutuhan meliputi kebutuhan perangkat keras maupun kebutuhan perangkat lunak. Perancangan sistem dibagi menjadi enam bagian berdasarkan fungsinya, yaitu: perancangan umum, perancangan REST *server*, perancangan *broker*, perancangan *subscriber middleware*, perancangan *database* dan perancangan *node sensor*. Berikut ini adalah gambar alur perancangan ditunjukkan oleh Gambar 3.3:



Gambar 3.3 Alur Perancangan Sistem

3.4 Implementasi

Implementasi dilakukan dengan mengacu pada tahap rekayasa kebutuhan dan perancangan. Implementasi REST *middleware publish-subscribe* meliputi:

1. Implementasi REST *server*, menjelaskan tentang pembuatan REST *server* dan proses translasi "*url to topic*", bagaimana mentranslasi HTTP request menjadi *publish* MQTT serta cara REST server menginput data ke dalam database
2. Implementasi *broker*, menjelaskan instalasi *mosquitto* sebagai MQTT *broker*
3. Implementasi *subscriber middleware*, menjelaskan proses penyimpanan data yang terkirim dari protocol MQTT ke Redis
4. Implementasi *database*, meliputi instalasi Redis dan MongoDB
5. Implementasi *node sensor*, berisi tentang kode program dari masing masing *platform*

3.5 Pengujian dan Analisis Hasil Pengujian

Dalam tahap ini dilakukan dengan mengumpulkan hasil uji dari parameter dengan cara menguji kinerja dan performa keseluruhan perangkat lunak yang sudah dikembangan sesuai dengan permasalahan yang telah dijabarkan pada bab sebelumnya. Pengujian fungsionalitas sistem pada penelitian ini memiliki beberapa skenario data yang digunakan untuk acuan pengujian perangkat lunak pada REST *middleware publish-subscribe*. Ada dua jenis pengujian yang akan dilakukan pada sistem yang telah dibangun yaitu pengujian fungsionalitas dan pengujian performa.

Pengujian fungsionalitas dilakukan dengan menggunakan beberapa scenario untuk menguji fitur-fitur yang pada perangkat lunak REST *middleware publish-subscribe* agar dapat berjalan dengan benar. Berikut ini tahap pengujian fungsionalitas atas berbagai skenario:

1. Peneliti menguji pengiriman data antara user ke perangkat *middleware*,
2. Peneliti menguji pengiriman *data sensor* dari *node sensor* ke perangkat *middleware*,
3. Peneliti menguji REST *middleware publish-subscribe* dalam proses melihat list *node sensor* beserta *live data sensor.*

Pengujian performa dilakukan dengan menggunakan beberapa skenario untuk mengetahui performa dari perangkat lunak REST *middleware publish-subscribe* yang dibangun. Berikut adalah tahap pengujian performa atas beberapa skenario:

1. Perbandingan performa mengirim *data sensor* ke *middleware* dengan ke *node sensor* secara langsung

   Peneliti mengukur berapa banyak *request* POST HTTP yang dapat ditangani *middleware* maupun *node sensor* ketika mengirimkan data ke *node sensor*. Dalam pengujian ini menggunakan dua parameter uji yaitu, *latency* dan *error rate*. Sedangkan penguji mencoba memberikan skenario uji sebanyak 50, 100, 150 pengguna dalam mengirim *request* HTTP secara bersamaan. Berikut ini adalah tabel uji pada proses POST HTTP baik pengujian terhadap *middleware* maupun ke *node sensor*:

Tabel 3.1 Tabel Uji Performansi HTTP POST

No	Jumlah User (*Threads*)	Average Latency	Average Error Rate
1	50	-	-
2	100	-	-
3	150	-	-

2. Perbandingan performa menerima data *sensor* dari *middleware* dengan ke *node sensor* secara langsung

   Peneliti mengukur berapa banyak *request* GET HTTP yang dapat ditangani *middleware* maupun *node sensor* ketika *user* mencoba mengakses *live data sensor*. Dalam pengujian ini menggunakan dua parameter uji yaitu, *latency* dan *error rate*. Sedangkan penguji mencoba memberikan skenario uji sebanyak 50, 100, 150 pengguna dalam mengirim *request* HTTP secara bersamaan.

Tabel 3.2 Tabel Uji Performansi HTTP GET

No	Jumlah User (*Threads*)	Average Latency	Average Error Rate
1	50	-	-
2	100	-	-
3	150	-	-

3. Perbandingan overhead kedua protokol pada proses melakukan koneksi dalam menerima perintah.

Tabel 3.3 Tabel Uji Overhead

No	Request Perintah (detik)	Besaran Overhead (HTTP)	Besaran Overhead (MQTT)
1	0.5	-	-
2	1	-	-
3	1.5	-	-
4	2	-	-

3.6 Kesimpulan dan Saran

Tahap pengambilan kesimpulan dapat dilakukan jika semua tahap sebelumnya telah selesai dibuat dan dilakukan. Kesimpulan dapat diambil melalui data pengujian terhadap sistem REST *middleware publish-subscribe* yang telah dibangun. Selain itu penulisan saran juga diperlukan sebagai acuan dalam memberikan pertimbangan mengenai pengembangan dan penelitian lanjut pada sistem REST *middleware publish-subscribe*.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN SISTEM

Pada bab ini dijelaskan mengenai kebutuhan fungsional dan non-fungsional yang wajib dipenuhi oleh sistem. Kebutuhan fungsional dapat berupa sebuah layanan, fitur dan proses yang disediakan bagi pengguna sistem. Kebutuhan non-fungsional berupa alat pendukung bagi pembuatan sistem. Kebutuhan yang telah didapat tersebut nantinya dikembangkan menjadi sebuah rancangan yang menjadikan landasan dalam proses implementasi sistem. Tahap perancangan tediri dari perancangan umum, perancangan REST *server*, perancangan *subscribe*r *middleware*, perancangan *database* dan perancangan *node sensor*.

4.1 Analisis Kebutuhan

Analisis kebutuhan adalah sebuah tahapan dalam mengumpulkan informasi tentang hal-hal yang dapat dilakukan oleh sistem dan kebutuhan sistem. Proses analisis kebutuhan dibagi menjadi dua yaitu analisis kebutuhan fungsional dan analisis kebutuhan non-fungsional.

4.1.1 Analisis Kebutuhan Fungsional

Kebutuhan fungsional berupa fitur yang harus dipenuhi agar sistem berjalan sesuai kebutuhan pengguna sistem. Berikut ini adalah kebutuhan fungsional dari sistem yang dibuat, dapat dilihat pada Tabel 4.1:

Tabel 4.1 Kebutuhan Fungsional

No	Kebutuhan Fungsional
1	Sistem dapat menerapkan translasi *"url-to-topic"* dari REST ke MQTT
2	Sistem dapat menyediakan layanan dalam mengakses data sensor pada tiap-tiap *node*
3	Sistem dapat menyediakan layanan registrasi *node sensor*
4	Sistem dapat menyediakan layanan penambahan *sensor*
5	Sistem dapat menampilkan *node sensor* yang terdaftar beserta *sensor*nya
6	Sistem dapat menyediakan layanan untuk memberikan perintah *sensor* yang terdapat pada *node sensor*

4.1.2 Analisis Kebutuhan Non-Fungsional

Kebutuhan non fungsional adalah batasan layanan atau fungsi yang ditawarkan oleh sistem, seperti batasan waktu, standarisasi dan lain-lain. Berikut ini adalah kebutuhan non fungsional dari sistem yang dibuat, seperti pada Tabel 4.2:

Tabel 4.2 Kebutuhan Non-Fungsional

No	Kebutuhan Non-Fungsional
1	Sistem dapat diakses baik dari perangkat desktop maupun perangkat mobile
2	Sistem dapat memberikan payload data dengan model data JSON

4.1.3 Analisis Kebutuhan Perangkat Keras

Kebutuhan perangkat keras dibutuhkan sebagai dasar utama dari pengembangan sistem yang dibangun. Perangkat keras yang terdapat pada pembangunan sistem ini dibagi menjadi tiga, pertama spesifikasi laptop, kedua spesifikasi *middleware* dan yang ketiga spesifikasi *node sensor* yang digunakan dalam pembangunan sistem REST *middleware publish-subscribe*. Berikut adalah spesifikasi perangkat keras yang dibutuhkan:

Tabel 4.3 Spesifikasi Laptop

Komponen	Spesifikasi
Processor	Intel(R) Core(TM) i5-4210U CPU @1.70Ghz
Memory (RAM)	8GB DDR3L
Harddisk	1000GB
Sistem Operasi	Windows 10 Pro

Tabel 4.4 Spesifikasi Middleware

Komponen	Spesifikasi
Processor	Intel(R) Core(TM) i5-4210U CPU @1.70Ghz
Memory (RAM)	1GB DDR3L
Harddisk	10GB
Sistem Operasi	Ubuntu 17.10 Desktop

Tabel 4.5 Spesifikasi Perangkat Node Sensor

Komponen	Spesifikasi
WEMOS D1	CPU : ESP-8266EX 80MHz/160MHz Flash : 4M bytes
*Node*MCU	CPU : ESP8266 Memory : 128 kBytes

	Storage : 4 Mbytes
Raspberry Pi 3 Model B	CPU :Quadcore BCM2837 64bit 1,2Ghz RAM : 1 GB Storage: 32 GB Sistem Operasi : Raspbian Jessie

4.1.4 Analisis Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak dibutuhkan sebagai dasar utama dari pengembangan sistem yang dibangun. Perangkat lunak yang terdapat pada pembangunan sistem ini dibagi menjadi enam bagian besar yaitu, pertama sebagai REST-Server, kedua sebagai sistem *database*, ketiga sebagai *code editor* dan *compiler*, keempat sebagai *broker*, kelima sebagai alat *testing* dan yang terakhir sebagai virtualisasi dari *middleware* yang digunakan dalam pembangunan sistem REST *middleware publish-subscribe*. Berikut adalah beberapa perangkat lunak yang dibutuhkan:

Tabel 4.6 Kebutuhan Perangkat Lunak

Komponen	Kegunaan
Python	Sebagai Bahasa pemrograman dalam membuat REST-Server, *subscriber middleware* dan *publisher* pada Raspberry PI
Flask	Library Python dalam membuat REST-Server
Redis	Sebagai *database* dalam menyimpan *live data sensor*
MongoDB	Sebagai *database* dalam menyimpan data *node sensor* yang terdaftar
Visual Studio Code	Sebagai code editor dalam membuat script REST-Server, *subscriber middleware* dan *publisher* pada Raspberry PI
ESPlorer	Sebagai *code editor* dan *compiler* dalam pembuatan *script* untuk perangkat NodeMCU
Arduino Ide	Sebagai *code editor* dan *compiler* dalam pembuatan *script* untuk perangkat Arduino
Mosquitto	Sebagai broker dalam mengimplementasikan arsitektur *publish-subscribe* antara *middleware* dengan *node sensor*
MQTTFX	Sebagai alat *testing* dalam melakukan *publish-subscribe* terhadap *broker*
Postman	Berperan sebagai user dalam mengakses REST-Server

Virtual Box	Sebagai virtualisasi perangkat middleware

4.2 Perancangan

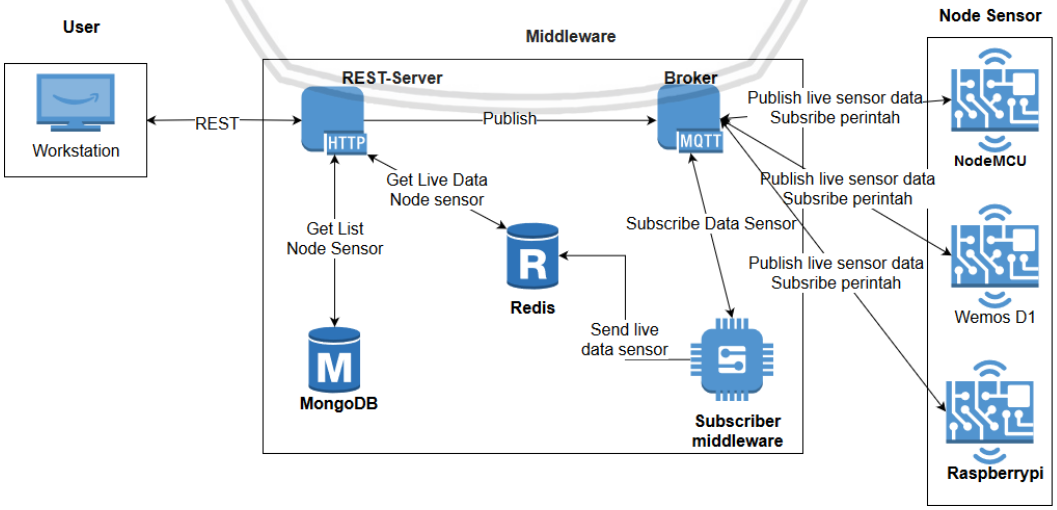
Pada bagian perancangan meliputi beberapa komponen penting seperti perancangan umum yang menjelaskan tentang bagaimana sistem bekerja yang terdiri dari perancangan REST server, perancangan *subscribe*r *middleware*, perancangan *database* dan perancangan *node sensor*.

4.2.1 Perancangan Umum

Pada penelitian kali ini kebutuhan utama terletak pada kebutuhan perangkat lunak REST *middleware publish-subscribe* yaitu Flask, Redis, MongoDB dan *mosquitto*. Dari ketiga perangkat lunak tersebut sudah harus terpasang pada sistem operasi ubuntu di *virtual machine*.

Dalam penelitian ini protokol MQTT digunakan sebagai protokol komunikasi antar mesin (*node sensor* dan *middleware*). Sedangkan pengguna tetap menggunakan protocol HTTP RESTful, format data yang digunakan dalam pengiriman data REST dapat berupa JSON. Pada tahap pengembangan, terdapat *IoT middleware* yang memiliki beberapa bagian utama terdiri dari REST *server*, *broker, Redis, MongoDB* dan *subscriber middleware*, sedangkan *node sensor* yang berfungsi sebagai *publisher* terhadap *broker*.

Pada Gambar 4.1 menjabarkan rancangan umum sistem yang dibangun. Terdapat 3 bagian penting dalam rancangan sistem ini yaitu *node sensor*, *middleware* dan *user*. *Node sensor* tersebut berupa *microcontroller* atau mikrokomputer yang dapat berupa Raspberry Pi, Wemos D1 dan *Node*MCU. Dalam sisi *middleware* menggunakan *virtual machine* yang mentranslasi perintah yang dikirimkan oleh pengguna berupa REST *Request* terhadap *middleware* lalu *middleware* meneruskan perintah ke *node sensor* dengan protocol MQTT.



Gambar 4.1 Perancangan Umum Sistem

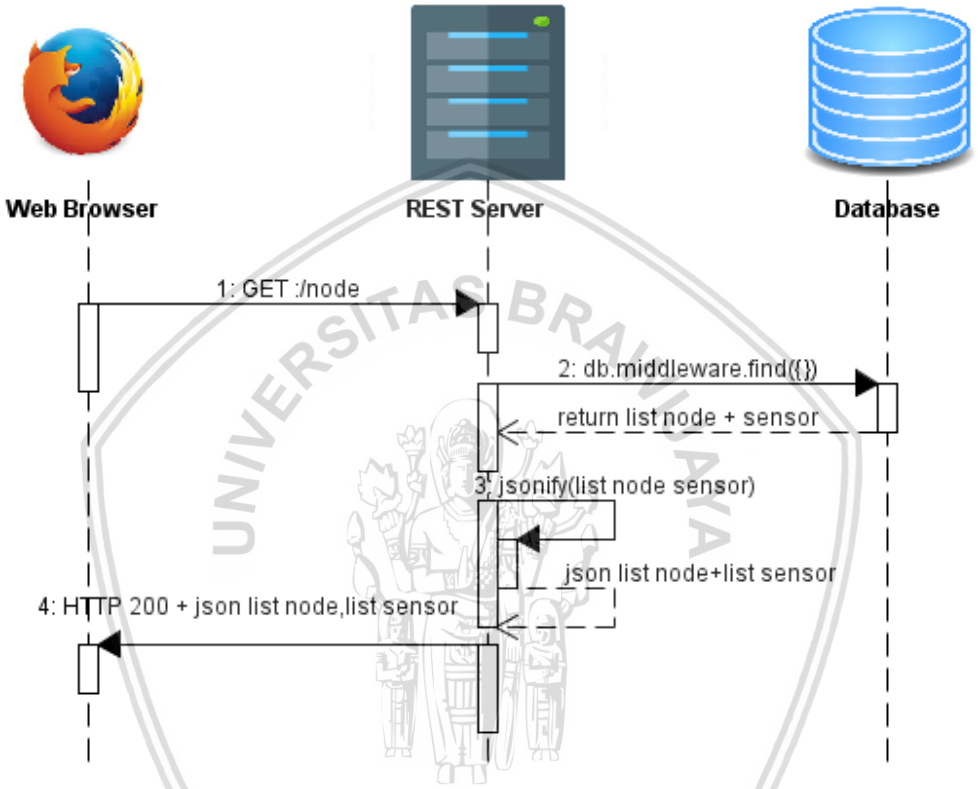
4.2.2 Perancangan REST Server

Perancangan REST server menyediakan akses terhadap fitur yang telah dilakukan pada Bab analisis kebutuhan. Dalam middleware yang akan dibangun ini, peneliti menyediakan sebuah API berupa REST server untuk mengakses data *sensor* dan melakukan pemberian perintah terhadap *sensor* pada *node sensor*. Dalam melakukan sebuah akses pada data *sensor* dapat dilakukan dengan cara melakukan *request* GET, sedangkan pada proses memberikan perintah terhadap *sensor* yaitu melakukan *request* POST. Di bawah ini dijelaskan mengenai URL yang dapat digunakan pengguna serta alur data yang terjadi pada server REST tersebut. Penjelasan mengenai daftar *request* yang didukung *middleware* dan informasi mengenai data yang didapatkan dari HTTP *response* terdapat pada tabel 4.7. Sedangkan alur data dan proses yang terjadi digambarkan pada Gambar 4.2 hingga 4.8.

Tabel 4.7 Daftar Request Middleware

No	Url	Method	Payload	Response
1	http://192.168.1.2:5000/node	GET	-	List *node sensor* berserta *sensor* yang terdaftar
2	http://192.168.1.2:5000/node/<name>	GET	-	List *node sensor* yang dipilih berserta *sensor* yang terdaftar
3	http://192.168.1.2:5000/node/<name>/sensor	GET	-	List *sensor* beserta live data *sensor* pada *node sensor* yang dipilih
4	http://192.168.1.2:5000/*node*/<name>/*sensor*/<*sensors*>	GET	-	List live data *sensor* pada *node sensor* yang dipilih
5	http://192.168.1.2:5000/add*node*	POST	JSON -node -label -topic	Memberikan pesan sukses ketika mendaftarkan *node sensor* beserta satu *sensor*
6	http://192.168.1.2:5000/add*sensor*	POST	JSON -node -label	Memberikan pesan sukses ketika menambahkan

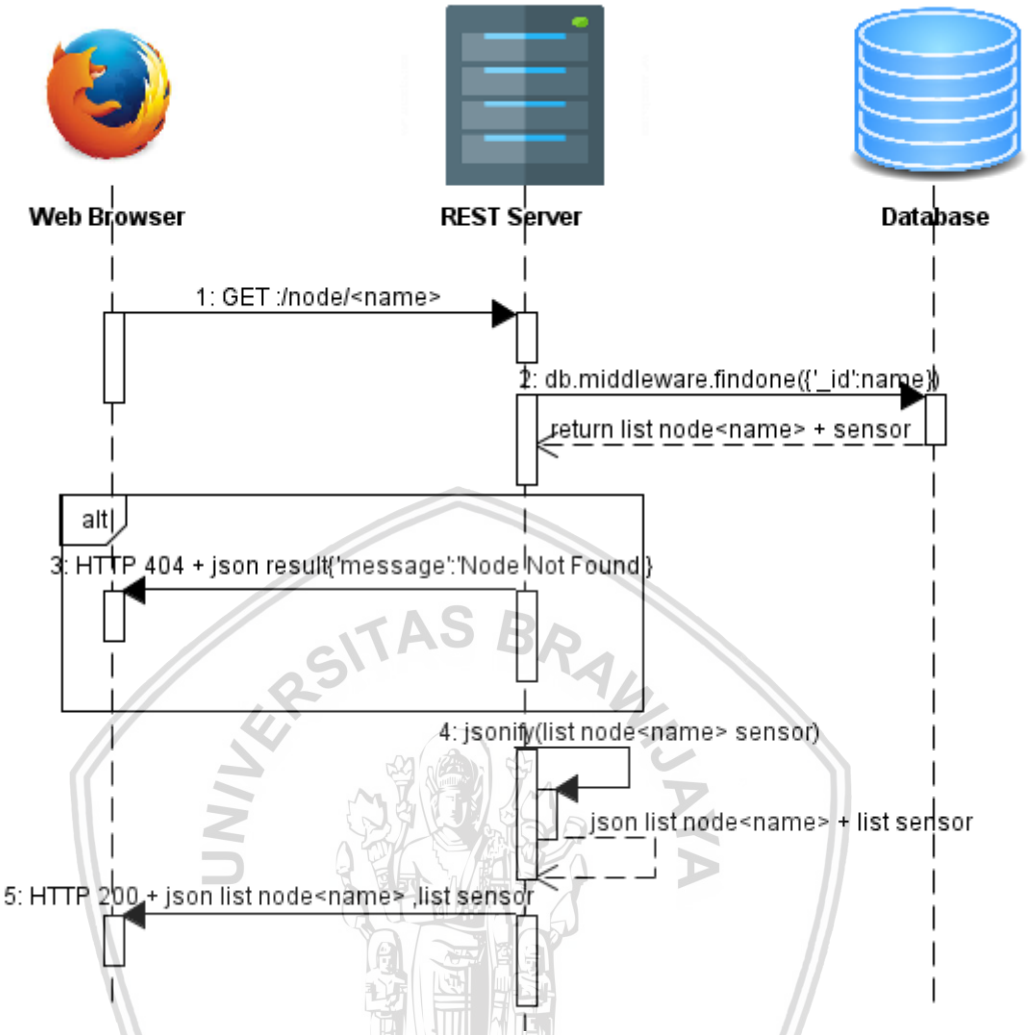
			-topic	*node sensor* beserta satu *sensor*
7	http://192.168.1.2:5000/*node*/<name>/*sensor*/<*sensors*>	POST	JSON - <sensors>	Memberikan pesan sukses publish



Gambar 4.2 Akses Data Node

Penjelasan dari Gambar 4.2 ialah sebagai berikut:

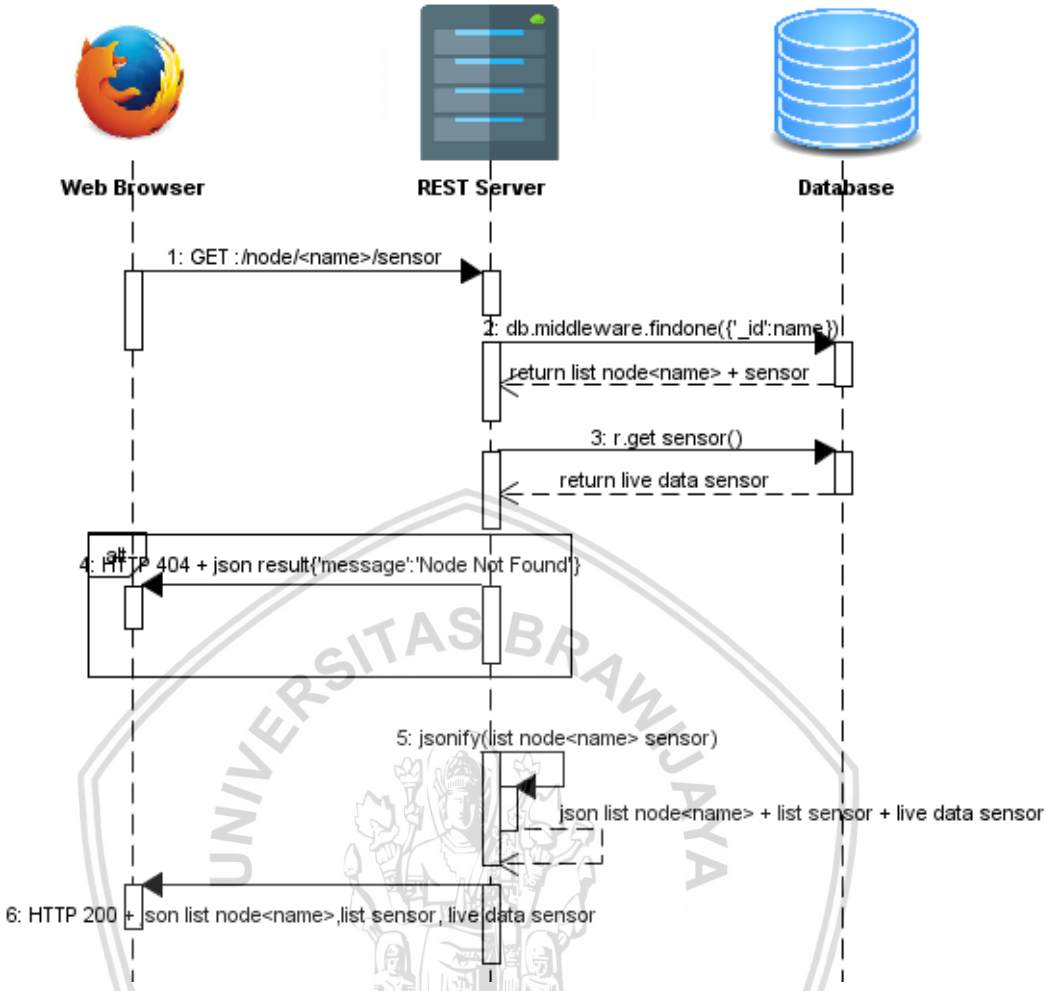
1) *Web browser* mengirimkan *request* HTTP dengan metode GET.
2) REST *server* memanggil fungsi `db.middleware.find({})` untuk mendapatkan data *node sensor* yang terdaftar. Kemudian *Database* mengembalikan data *node sensor* berserta *sensor*nya yang terdaftar.
3) REST *server* melakukan proses *parsing list node* dan *sensor* dari dokumen MongoDB menjadi json.
4) REST *server* mengirimkan response HTTP status 200 beserta data list *node* dan *sensor* yang telah ubah formatnya menjadi json.



Gambar 4.3 Akses Data Node Berdasarkan Nama Node

Penjelasan dari Gambar 4.3 ialah sebagai berikut:

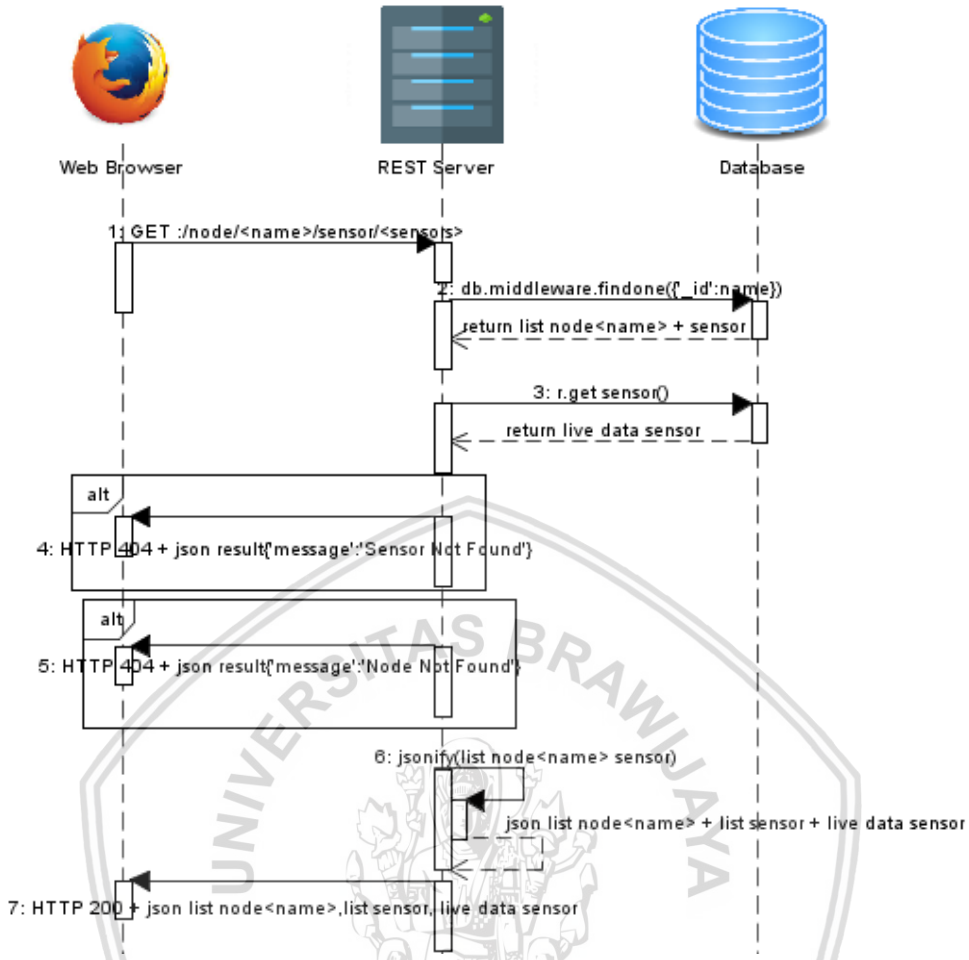
1) Web browser mengirimkan *request* HTTP dengan metode GET.
2) REST server memanggil fungsi `db.middleware.findone({})` untuk mendapatkan data *node sensor* dengan parameter name sebagai id dalam *database* MongoDB. Kemudian *database* mengembalikan data *node sensor* berserta *sensor*nya yang terdaftar.
3) Alternatif apabila data *node* pada *database* tidak ditemukan maka REST server mengirimkan response HTTP status 404 beserta json dengan message "*Node* Not Found".
4) REST server melakukan proses *parsing* list *sensor* dari dokumen MongoDB menjadi json.
5) REST server mengirimkan response HTTP status 200 beserta data list *sensor* yang telah ubah formatnya menjadi json.



Gambar 4.4 Akses Data Sensor Berdasarkan Nama Node

Penjelasan dari Gambar 4.4 ialah sebagai berikut:

1) *Web browser* mengirimkan *request* HTTP dengan metode GET.
2) REST *server* memanggil fungsi `db.middleware.findone({})` untuk mendapatkan data *node sensor* dengan parameter *name* sebagai id dalam *database* MongoDB. Kemudian *database* mengembalikan data *node sensor* berserta *sensor*nya yang terdaftar.
3) REST *server* memanggil fungsi `r.get sensor()` untuk mendapatkan *live data sensor* dalam *database* Redis. Kemudian Redis mengembalikan *live data sensor*.
4) Alternatif apabila data *node* pada *database* tidak ditemukan maka REST *server* mengirimkan response HTTP status 404 beserta json dengan *message* "*Node Not Found*".
5) REST *server* melakukan proses *parsing* list *sensor* dan live data *sensor* dari dokumen MongoDB menjadi json.
6) REST *server* mengirimkan response HTTP status 200 beserta data list *sensor* dan live data *sensor* yang telah ubah formatnya menjadi json.

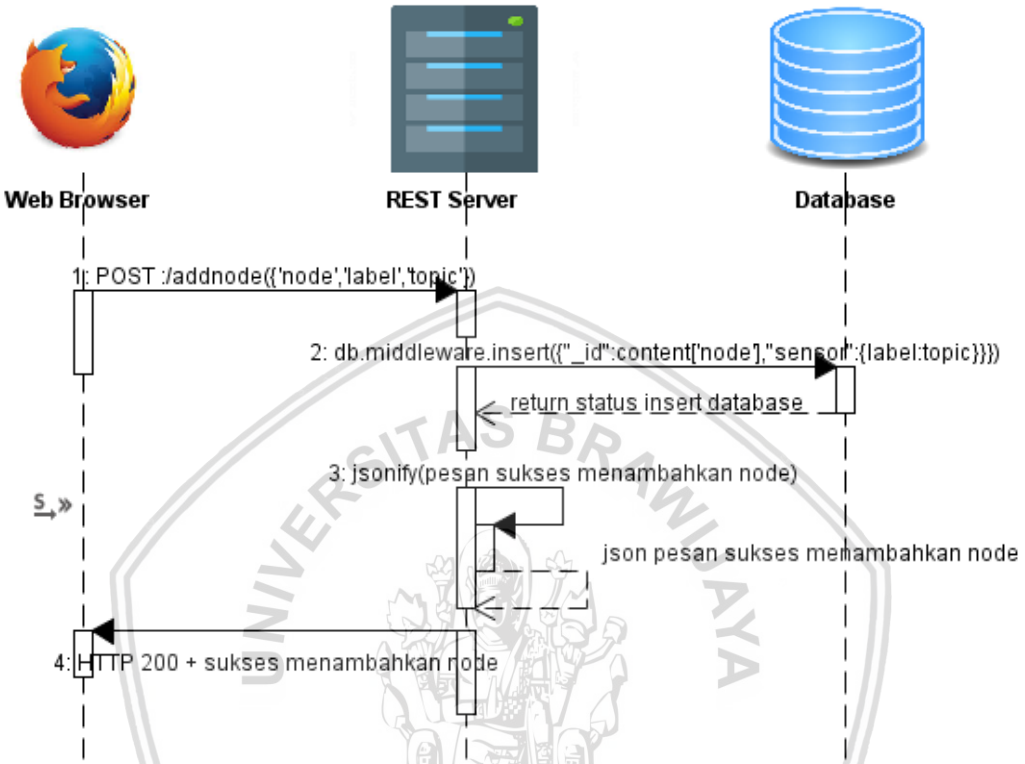


Gambar 4.5 Akses Data Satu Sensor Berdasarkan Nama Node

Penjelasan dari Gambar 4.5 ialah sebagai berikut:

1) Web browser mengirimkan *request* HTTP dengan metode GET.
2) REST server memanggil fungsi `db.middleware.findone({})` untuk mendapatkan data *node sensor* dengan parameter *name* sebagai id dalam *database* MongoDB. Kemudian *database* mengembalikan data *node sensor* berserta *sensor*nya yang terdaftar.
3) REST server memanggil fungsi `r.get sensor()` untuk mendapatkan *live data sensor* dalam *database* Redis. Kemudian Redis mengembalikan *live data sensor*.
4) Alternatif apabila data *sensor* pada *database* tidak ditemukan maka REST server mengirimkan response HTTP status 404 beserta json dengan message "*Sensor Not Found*".
5) Alternatif apabila data *node* pada *database* tidak ditemukan maka REST server mengirimkan response HTTP status 404 beserta json dengan message "*Node Not Found*".
6) REST server melakukan proses *parsing* list *sensor* dan *live data sensor* dari dokumen MongoDB menjadi json.

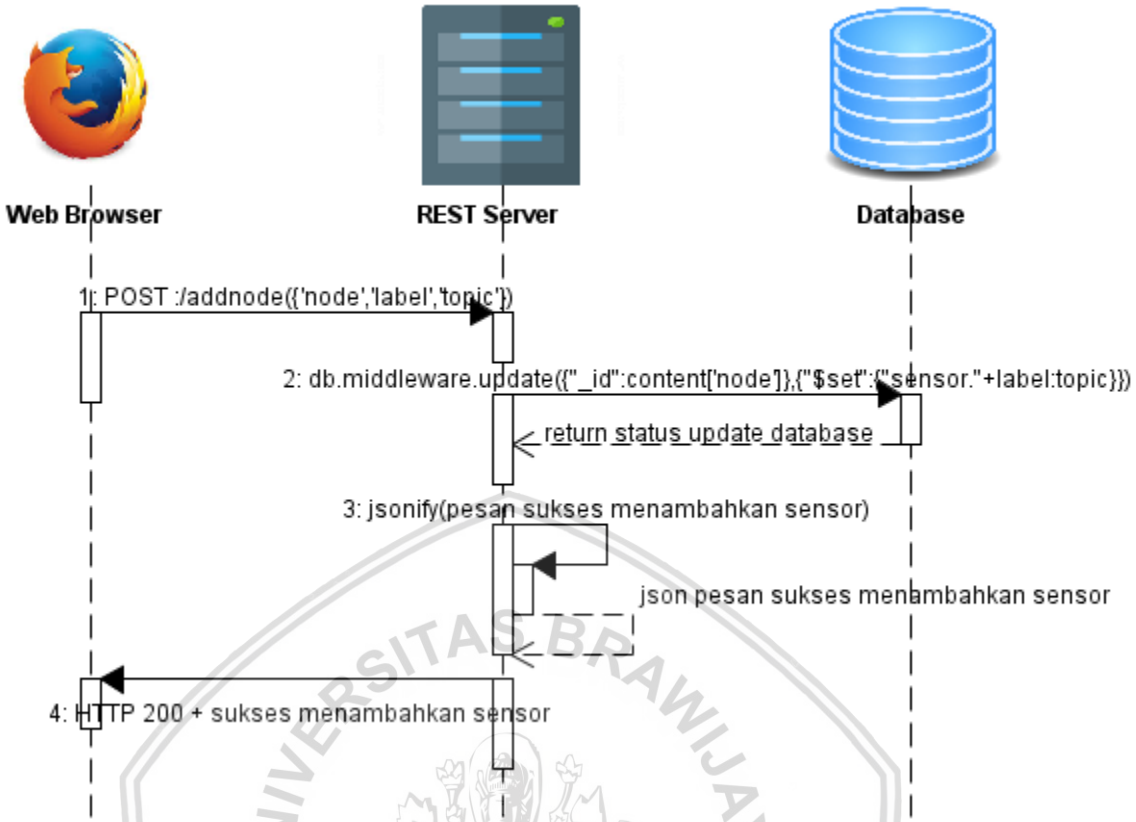
7) REST server mengirimkan response HTTP status 200 beserta data list *sensor* dan *live data sensor* yang telah ubah formatnya menjadi json.



Gambar 4.6 Akses Data Registrasi Node Sensor

Penjelasan dari Gambar 4.6 ialah sebagai berikut:

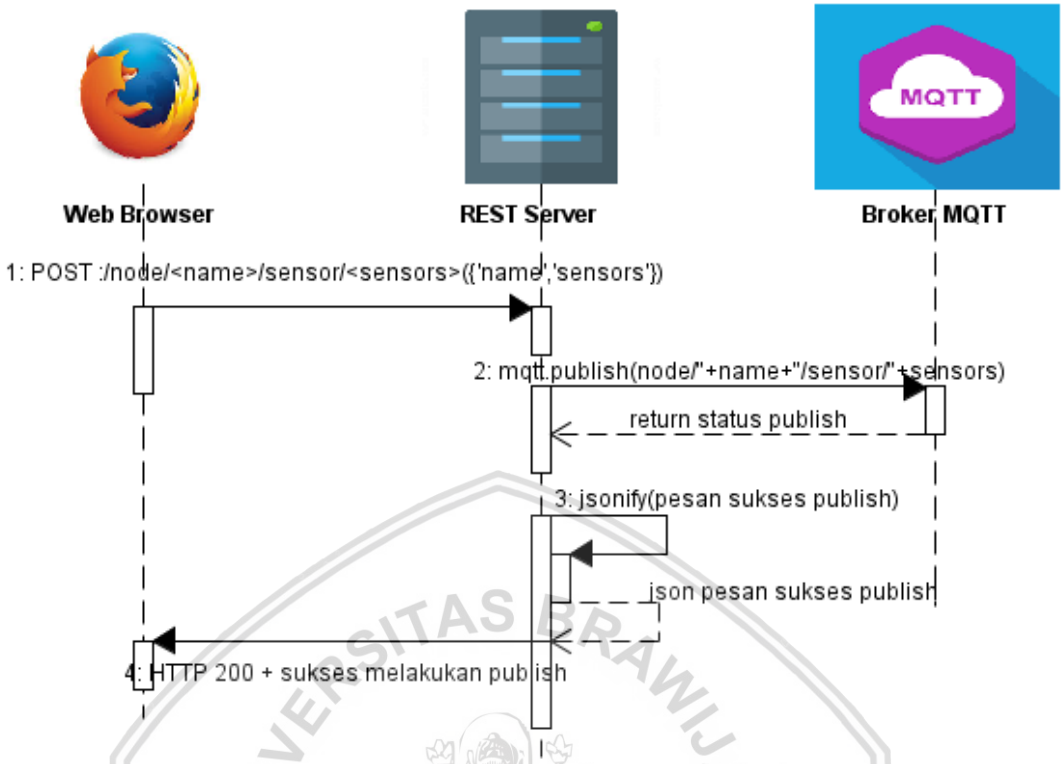
1) Web browser mengirimkan *request* HTTP dengan metode POST dengan parameter JSON Object yang didalamnya terdapat *field node*, *label* dan *topic*.
2) REST *server* memanggil fungsi `db.middleware.insert({})` untuk memberikan proses *input* pada *database* MongoDB dengan parameter *node*, *label* dan *topic*. Kemudian *database* mengembalikan status *input* dari data yang telah disebutkan tadi.
3) REST *server* melakukan proses *parsing* pesan sukses menambahkan *node*.
4) REST *server* mengirimkan response HTTP status 200 beserta data status pesan sukses dari proses *input node*.



Gambar 4.7 Akses Data Registrasi Sensor

Penjelasan dari Gambar 4.7 ialah sebagai berikut:

1) Web browser mengirimkan *request* HTTP dengan metode POST dengan parameter json Object yang didalamnya terdapat *field node*, *label* dan *topic*.
2) REST *server* memanggil fungsi `db.middleware.update({})` untuk memberikan proses *input* pada *database* MongoDB dengan parameter *node*, *label* dan *topic*. Kemudian *database* mengembalikan status *input* dari data yang telah disebutkan tadi.
3) REST *server* melakukan proses *parsing* pesan sukses menambahkan *sensor*.
4) REST *server* mengirimkan response HTTP status 200 beserta data status pesan sukses dari proses *input sensor*.



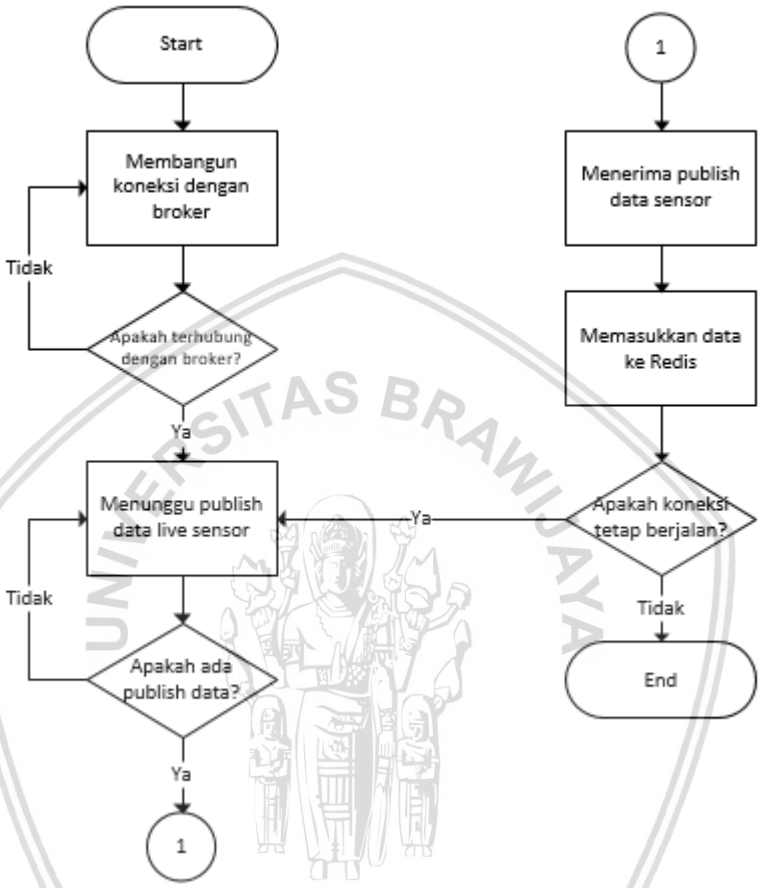
Gambar 4.8 Akses Data Publish Perintah Node Sensor

Penjelasan dari Gambar 4.8 ialah sebagai berikut:

1) Web browser mengirimkan *request* HTTP dengan metode POST dengan parameter JSON Object yang didalamnya terdapat field <*sensors*>, field *sensor* dapat berubah sesuai dengan *sensor* yang yang diberi perintah publish. Contoh jika ingin *publishing* pada *node sensor* dengan *sensor* temperature maka *field sensors* berubah menjadi "*temperature*".
2) REST *server* memanggil fungsi `mqtt.publish()` kepada *broker* MQTT untuk proses *publish* pada *node sensor* dengan parameter url sebagai *topic* dan field <*sensors*> sebagai data yang dikirimkan ke *node sensor*. Kemudian *broker* mengembalikan status *publish*.
3) REST *server* melakukan proses *parsing* pesan sukses *publish*.
4) REST *server* mengirimkan response HTTP status 200 beserta data status pesan sukses *publish*.

4.2.3 Perancangan Subscriber Middleware

Subscriber middleware ditulis dalam bahasa Python. Tugas bagi *subscriber middleware* di sini adalah meneruskan data *live sensor* ke dalam *database Redis* dengan pola *key* dan *value*. Pada *key* Redis diisi dengan topik MQTT dan *value* diisi dengan nilai data *sensor* tersebut.



Gambar 4.9 Alur Kerja Subsriber Middleware

Pada Gambar 4.9 menggambarkan alur kerja *flowchart* dari *subscriber middleware*. Jika *subscriber middleware* berjalan, dimulai membangun koneksi dengan *Broker* MQTT yang terdapat di dalam *middleware* apabila sudah terhubung maka selanjutnya melakukan proses *subscribe* pada *topicwildcard* "*node*/#". Setelah proses *subscribe* selesai maka *subscriber middleware* menunggu publish data *live sensor* dari *node sensor*, jika terdapat *publish data sensor* maka *subscriber middleware* akan menerima data dan meneruskan data ke *database* Redis.

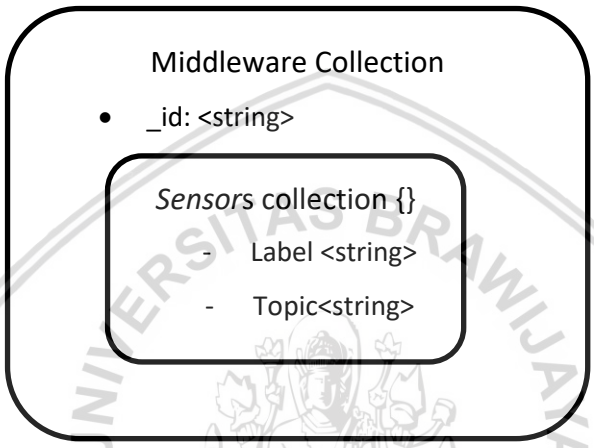
4.2.4 Perancangan Database

Pada perancangan *database* dibagi menjadi dua yakni perancangan *database* MongoDB dan perancangan *database* Redis. Perancangan *database* MongoDB dikhususkan dalam proses penyimpanan list *node sensor* yang terdaftar beserta

*sensor*nya. Sedangkan pada perancangan *database* Redis dikhususkan dalam proses penyimpanan data *live sensor*.

4.2.4.1 Perancangan Database MongoDB

Pada perancangan *database* MongoDB yang berbasis dokumen tentunya memerlukan penyesuaian terhadap data yang akan disimpan. Dalam proses identifikasi diperlukannya satu dokumen yang bernama *middleware*. Dokumen *middleware* menyimpan informasi mengenai list perangkat *IoT* beserta *sensor* yang dimilikinya. Untuk penjelasan lebih detail dapat dilihat pada Gambar 5.10.



Gambar 4.10 Skema Collection MongoDB

Pada Gambar 4.10 terdapat dua field utama, yang pertama "_id" berfungsi sebagai *primary key* pada *database* MongoDB yang dapat diisi tipe data *string*. Pada *field* kedua yang merupakan *embedded* document, di mana sebuah *record* pada MongoDB dapat memiliki beberapa record, diasumsikan dalam satu perangkat *IoT* dapat memiliki beberapa *sensor* yang terhubung. Oleh karena itu untuk mengurangi redundansi data maka peneliti menggunakan *embedded document* pada *sensor collection*.

4.2.4.2 Perancangan Database Redis

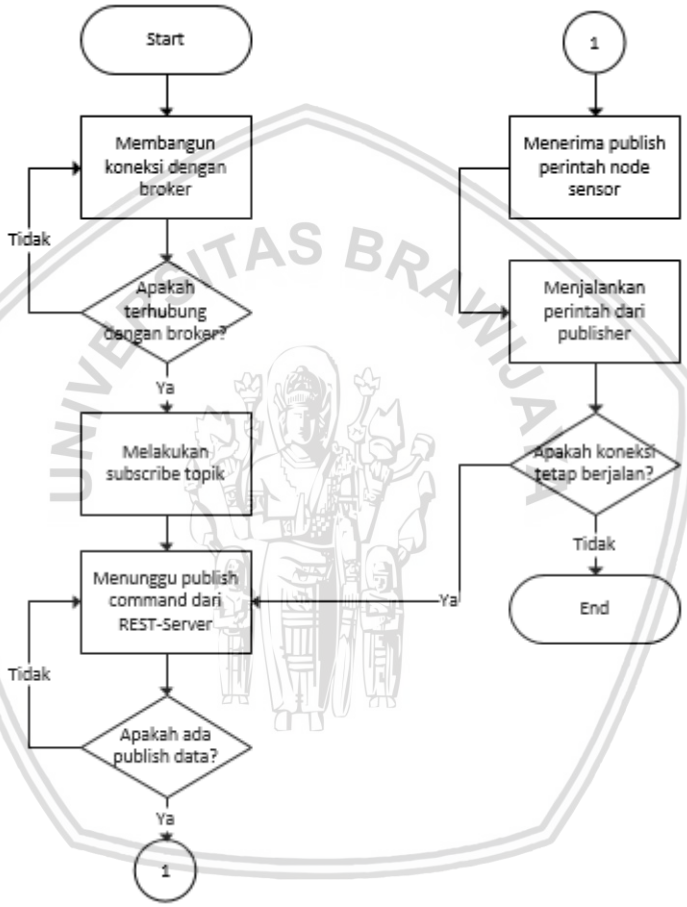
Pada perancangan *database* Redis tidak serumit MongoDB dikarenakan Redis berbasis *key* dan *value* saja. Kami menggunakan masing-masing *topic* mqtt sebagai *key* dan data sensor sebagai *value*nya. Untuk contoh mengenai struktur *database* Redis dapat dilihat pada Tabel 4.13.

Tabel 4.8 Struktur Database Redis

Key	value
node/*node*/*sensors*/temperature	50
node/*node*2/*sensor*/*humidity*	70

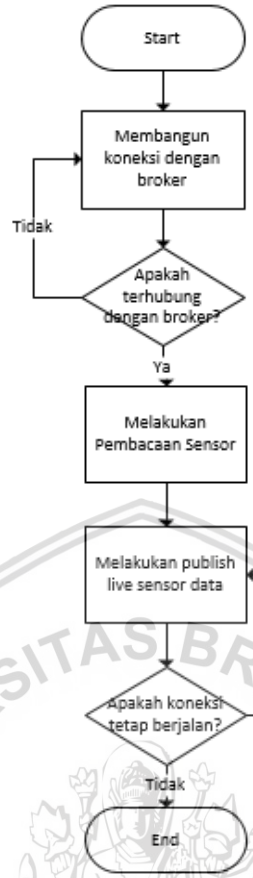
4.2.5 Perancangan Node Sensor

Pada dasarnya *node sensor* dapat diasumsikan sebagai pengirim data *sensor* sekaligus sebagai penerima perintah MQTT yang sebelumnya adalah perintah REST yang ditranslasi. Terdapat tiga perangkat *IoT* yang digunakan dalam penelitian ini dan juga terdapat dua *sensor* yang digunakan ialah *sensor* led dan *sensor* dht11 yang dapat mengukur temperatur dan kelembapan udara. Untuk lebih lanjut mengenai bagaimana *node sensor* mengirim data dan menerima perintah dapat dilihat pada Gambar 4.11 dan 4.12. Sedangkan perancangan perangkat dapat dilihat pada Gambar 4.13, 4.14 dan 4.15.



Gambar 4.11 Alur Kerja Subscriber pada Node Sensor

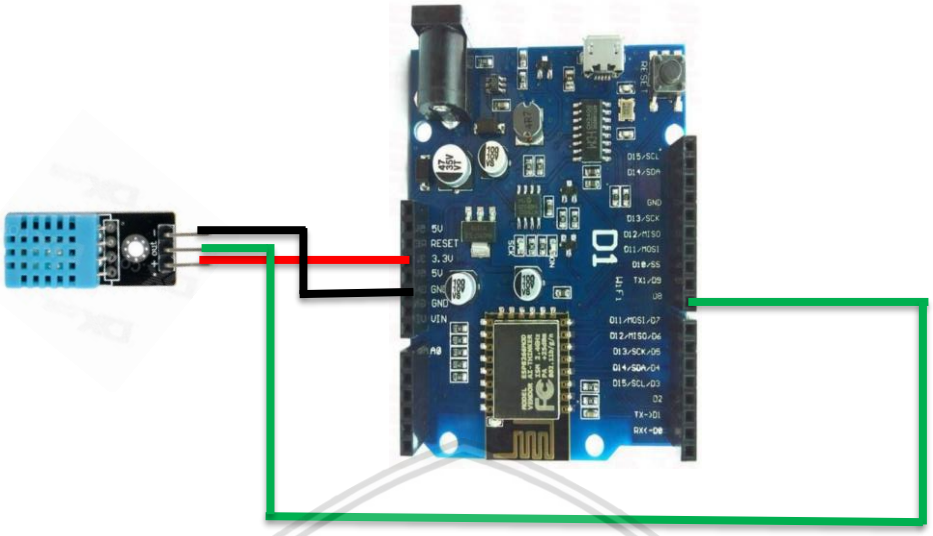
Pada Gambar 4.11 menggambarkan alur kerja flowchart dari *Subscribe*r pada *Node sensor*. Diasumsikan perangkat *node sensor* sudah terhubung pada jaringan, lalu *node sensor* membangun koneksi dengan *broker* MQTT yang terdapat pada *middleware*. Apabila sudah terhubung, maka selanjutnya proses *subscribe* pada topik yang ditentukan. Setelah proses *subscribe* selesai maka *node sensor* menunggu publish data dari *broker* yang sebelumnya dipicu oleh REST *Server*, jika terdapat data yang dipublish oleh *broker* selanjutnya *node sensor* akan menerima data dan melakukan perintah sesuai yang didesain pada perangkat *node sensor*.



Gambar 4.12 Alur Kerja Publisher pada Node Sensor

Pada Gambar 4.12 menggambarkan alur kerja *flowchart* dari *publisher* pada *node sensor*. Diasumsikan perangkat *node sensor* sudah terhubung pada jaringan, lalu *node sensor* membangun koneksi dengan *broker* MQTT yang terdapat pada *middleware*. Apabila sudah terhubung, maka selanjutnya proses pembacaan *sensor* dan melakukan *publish data sensor* pada topik yang ditentukan.

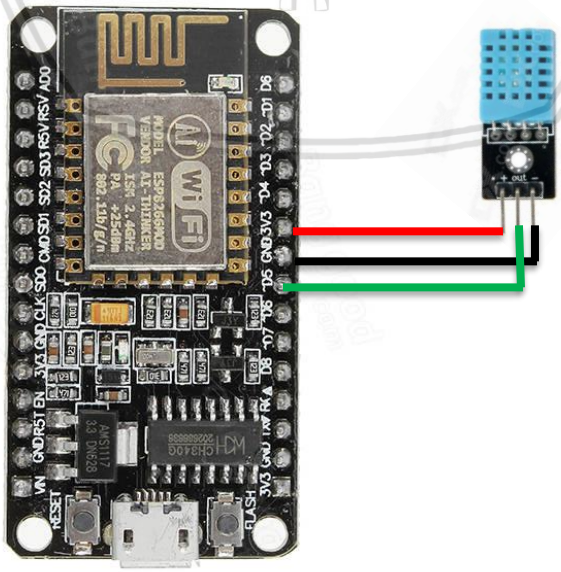
4.2.5.1 Perancangan Node Sensor *Arduino*



Gambar 4.13 Perancangan Node Sensor Arduino

Pada Gambar 4.13 menggambarkan perancangan *node sensor* Arduino, *sensor* yang digunakan adalah *sensor* led dan dht11 di mana *sensor* tersebut dapat mengukur suhu ruangan dan tingkat kelembapannya. Pada *sensor* led kami menggunakan led *onboard*, sedangkan *sensor* dht11 dihubungkan dengan port gpio. *Sensor* dht11 yang digunakan adalah *sensor* dht11 yang memiliki 3 pin, antara lain pin vcc ditandai dengan warna merah disambungkan dengan gpio 3.3 volt, pin *ground* ditandai dengan warna hitam disambungkan dengan pin gnd sedangkan pin sinyal ditandai dengan warna hijau disambungkan ke pin D8.

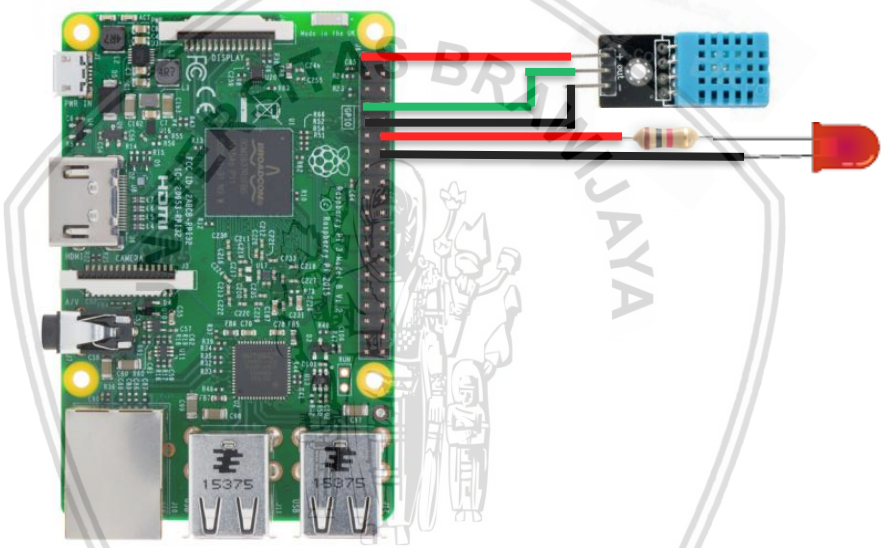
4.2.5.2 Perancangan Node Sensor *NodeMCU*



Gambar 4.14 Perancangan Node Sensor NodeMCU

Pada Gambar 4.14 menggambarkan perancangan *node sensor* NodeMCU, *sensor* yang digunakan adalah *sensor* led dan dht11 di mana *sensor* tersebut dapat mengukur suhu ruangan dan tingkat kelembapannya. Pada *sensor* led kami menggunakan led onboard, sedangkan *sensor* dht11 dihubungkan dengan port gpio. *Sensor* dht11 yang digunakan adalah *sensor* dht11 yang memiliki 3 pin, antara lain pin vcc ditandai dengan warna merah disambungkan dengan gpio 3.3 volt, pin ground ditandai dengan warna hitam disambungkan dengan pin gnd sedangkan pin sinyal ditandai dengan warna hijau disambungkan ke pin D6.

4.2.5.3 Perancangan Node Sensor RaspberryPI



Gambar 4.15 Perancangan Node Sensor RaspberryPI

Pada Gambar 4.15 menggambarkan perancangan *node sensor* RaspberryPI, *sensor* yang digunakan adalah *sensor* led dan dht11 di mana *sensor* tersebut dapat mengukur suhu ruangan dan tingkat kelembapannya. Dikarenakan tidak terdapat led bawaan pada raspberrypi maka kami menggunakan led beserta resistor tambahan. Baik *sensor* led maupun *sensor* dht11 dihubungkan dengan port gpio. *Sensor* dht11 yang digunakan adalah *sensor* dht11 yang memiliki 3 pin, antara lain pin vcc ditandai dengan warna merah disambungkan dengan gpio 3.3 volt, pin ground ditandai dengan warna hitam disambungkan dengan pin gnd sedangkan pin sinyal ditandai dengan warna hijau disambungkan ke pin gpio4.

BAB 5 IMPLEMENTASI

Pada bagian implementasi dijelaskan mengenai implementasi sistem yang mengacu pada bab analisis kebutuhan dan perancangan. Pada proses implementasi meliputi beberapa komponen penting seperti implementasi REST server, implementasi *broker*, implementasi *subscribe*r *middleware*, implementasi *database* dan implementasi *node sensor*.

5.1 Implementasi REST Server

Pada implementasi REST server, diperlukan proses update dan upgrade pada sistem linux untuk mendapatkan versi aplikasi yang terbaru. Aplikasi REST server yang digunakan pada penelitian ini adalah *framework* Flask yang berjalan pada Python. Berikut ini adalah kode yang perlu dijalankan agar aplikasi Flask terpasang pada *middleware*.

```
1  sudo apt-get update
2  sudo apt-get upgrade
3  sudo apt-get install Python
4  sudo apt-get install Python-pip
5  sudo pip install Flask
```

Setelah melakukan instalasi aplikasi yang dibutuhkan, maka perlu membuat script REST *server* yang nantinya berfungsi sebagai penerjemah dari protocol HTTP ke MQTT. Berikut contoh script REST server *middleware*:

```
1   r = Redis.StrictRedis(host='localhost',port=6379, db=0)
2   app = Flask(__name__)
3   app.config['MQTT_BROKER_URL'] = '127.0.0.1'
4   app.config['MQTT_BROKER_PORT'] = 1883
5   app.config['MQTT_USERNAME'] = ''
6   app.config['MQTT_PASSWORD'] = ''
7   app.config['MQTT_KEEPALIVE'] = 5
8   app.config['MQTT_TLS_ENABLED'] = False
9   app.config['MONGO_DBNAME'] = 'middleware'
10  mongo = PyMongo(app, config_prefix='MONGO')
11  mqtt = Mqtt(app)
12
13  @app.route('/addnode', methods=['GET','POST'])
14  def addnode():
15      if request.method=='POST':
16          content = request.get_json()
17          label = content['label']
18          topic = content['topic']
19
20  mongo.db.middleware.insert({"_id":content['node'],"sensors":{label:t
21  opic}})
22         output={'message':'Success Adding Node :'+content['node']}
23          return jsonify({'result':output})
24
25  @app.route('/addsensor', methods=['GET','POST'])
26  def addsensor():
27      if request.method=='POST':
28          content = request.get_json()
29          label = content['label']
30          topic = content['topic']
31
```

```
mongo.db.middleware.update({"_id":content['node']},{"$set":{"sensors
."+label:topic}})
       output={'message':'Success Adding Sensor : '+label+ ' at ' +
content['node'] }
        return jsonify({'result':output})


@app.route('/node',methods=['GET'])
def shownode():
    platform = mongo.db.middleware
    output = []
    for h in platform.find():
        output.append(h)
    return jsonify({'result':output})

@app.route('/node/<name>',methods=['GET'])
def showsensor(name):
    platform = mongo.db.middleware
    h = platform.find_one({'_id':name})
    output = []
    if h:
        output.append(h)
    else:
        output={'message':'Node Not Found'}
       response = jsonify({'result':output})
       response.status_code=404
        return response
    return jsonify({'result':output})

@app.route('/node/<name>/sensor',methods=['GET'])
def checknodesensor(name):
    if request.method=='GET':
        platform = mongo.db.middleware
        h = platform.find_one({'_id':name})
        output = []
        listsensor=[]

        status=""
        if h:
            output.append(h)
            dicthasil={}
            dicthasil.update(output[0]['sensors'])
            for key, value in dicthasil.iteritems():

                if r.get("node/"+name+"/"+value+"/status"):
                    data=r.get("node/"+name+"/"+value+"/status")
                    stats={key:data}
                    listsensor.append(dict(stats))
                else:
                  stats={key:'Sensor Not Found'}
                  listsensor.append(dict(stats))
        else:
            output={'message':'Node Not Found'}
           response = jsonify({'result':output})
           response.status_code=404
            return response
        return jsonify({'result':listsensor})

@app.route('/node/<name>/sensor/<sensors>',methods=['GET','POST'])
def pubsubsensor(name, sensors):
    if request.method=='GET':
        platform = mongo.db.middleware
        h = platform.find_one({'_id':name})
        output = {}
        topic="node/"+name+"/sensor/"+sensors+"/status"
```

```
97          status=""
98          if h:
99              output.update(h)
100             if r.get(topic):
101                 status = r.get(topic)
102             else:
103                 output={'message':'Sensor Not Found'}
104                    response = jsonify({'result':output})
105                    response.status_code=404
106                 return response
107         else:
108                output={'message':'Node Not Found'}
109                response = jsonify({'result':output})
110                response.status_code=404
111             return response
112           output={'status':status}
113         return jsonify({'result':output})
114     elif request.method=='POST':
115         content = request.get_json()
116         topic="node/"+name+"/sensor/"+sensors
117         message=content[sensors]
118         mqtt.publish(topic, message)
119         output={'message':'Publish
120 success','topic':topic,'message':message}
121       return jsonify({'result':output})
122
123 if __name__ == '__main__' :
124     app.run(host='0.0.0.0',   debug   =   False,   use_reloader=True,
125 threaded = True)
```

Penjelasan dari potongan kode program diatas adalah:

1. Baris ke 1 melakukan konfigurasi koneksi pada redis.
2. Baris ke 2-8 melakukan konfigurasi flask-mqtt yang yang berisi alamat broker, port, username, password dan opsi lainnya.
3. Baris ke 9 melakukan konfigurasi koneksi pada database mongodb dengan collection "middleware".
4. Baris 10-11 melakukan inisiasi pada koneksi database mongodb dan koneksi dengan broker mqtt.
5. baris ke 13-23 melakukan inisiasi pada url /addnode dimana hanya dapat diakses dengan HTTP POST dengan payload data json, json yang dikirim pengguna akan disimpan pada variable content lalu melakukan inisiasi pada variabel label dan topic dimana pada variable tersebut mengacu pada proses decode json yang memiliki key "label" dan "topic". Setelah itu melakukan proses insert data node mongodb dengan parameter "_id" dan "sensors" dengan payload json yang dikirim pengguna dan melakukan pengembalian response dengan json bahwa telah sukses menambahkan node.
6. Baris ke 25-38 melakukan inisiasi pada url /addsensor dimana hanya dapat diakses dengan HTTP POST dengan payload data json, json yang dikirim pengguna akan disimpan pada variable content lalu melakukan inisiasi pada variabel label dan topic dimana pada variable tersebut mengacu pada proses decode json yang memiliki key "label" dan "topic". Setelah itu melakukan proses insert data sensor

mongodb dengan parameter "_id" dan "sensors" dengan payload json yang dikirim pengguna dan melakukan pengembalian response dengan json bahwa telah sukses menambahkan sensor.

7. Baris ke 39-45 melakukan inisiasi pada url /node dimana hanya dapat diakses dengan HTTP GET, lalu melakukan select data dan disimpan dalam variable output. Setelah itu melakukan pengembalian response dari variable output tadi diparsing menjadi json.
8. Baris ke 47-59 melakukan inisiasi pada url /node/<name> dimana hanya dapat diakses dengan HTTP GET, lalu melakukan select satu data pada mongoDB dengan id:name yang disisipkan pada url dan disimpan dalam variable output. Setelah itu melakukan pengembalian response dari variable output tadi diparsing menjadi json.
9. Baris ke 47-59 melakukan inisiasi pada url /node/<name> dimana hanya dapat diakses dengan HTTP GET, lalu melakukan select satu data pada mongoDB dengan id:name yang disisipkan pada url dan disimpan dalam variable output. Setelah itu melakukan pengembalian response dari variable output tadi diparsing menjadi json.
10. Baris ke 61-88 melakukan inisiasi pada url /node/<name>/sensor dimana hanya dapat diakses dengan HTTP GET, lalu melakukan select satu data pada mongoDB dengan id:name yang disisipkan pada url dan disimpan dalam variable output. Setelah itu melakukan select data sensor pada redis dengan key yang terdapat pada variable output dan disimpan dalam variabel listsensor dan melakukan pengembalian response dari variable listsensor tadi diparsing menjadi json.
11. Baris ke 90-121 melakukan inisiasi pada url /node/<name>/sensor/<sensors> dimana dapat diakses dengan HTTP GET maupun dengan HTTP POST, ketika diakses dengan method GET maka akan melakukan select satu data sensor pada redis dengan key dari variabel topic dan mengembalikan response dari variabel output yang diparsing menjadi json. Apabila diakses menggunakan method POST maka REST server melakukan publish MQTT terhadap broker dengan topic yang terdapat dalam variabel topic dan payload data yang terdapat pada variable message. Setelah itu melakukan pengembalian response dengan json bahwa telah sukses melakukan publish data.
12. Baris ke 123-125 merupakan settingan tambahan pada framework flask, seperti berjalan pada ip 0.0.0.0, debug mode off, penggunaan thread dan opsi pada penggunaan reloader.

5.2 Implementasi Broker

Pada implementasi *broker*, diperlukan proses update dan upgrade pada sistem linux untuk mendapatkan versi aplikasi yang terbaru. Aplikasi *broker* yang

digunakan pada penelitian ini adalah *mosquitto*. Berikut ini adalah kode yang perlu dijalankan agar aplikasi *mosquitto* terpasang pada *middleware*.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mosquitto
```

5.3 Implementasi Subscriber Middleware

Pada implementasi *subscribe*r *middleware*, diperlukan proses update dan upgrade pada sistem linux untuk mendapatkan versi aplikasi yang terbaru. Pada *subscribe*r *middleware* ini menggunakan Bahasa pemrograman Python, oleh karena itu dibutuhkan aplikasi Python dan beberapa library pendukung yang dibutuhkan *subscribe*r dari menerima data hingga memasukkannya kedalam *database* Redis. Berikut ini adalah kode yang perlu dijalankan agar aplikasi Python dan librarynya terpasang pada *middleware*.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install Python
sudo apt-get install Python-pip
sudo pip install paho-mqtt
sudo pip install Redis
```

Setelah melakukan instalasi aplikasi yang dibutuhkan, maka perlu membuat script yang nantinya berfungsi sebagai penerima data dari *broker* lalu memasukkannya ke dalam *database* Redis. Berikut contoh script *subscribe*r *middleware*:

```
r = Redis.StrictRedis(host='localhost',port=6379, db=0)

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker")
        global Connected
        Connected = True
    else:
        print("Connection failed")

def on_message(client, userdata, message):
    print("Message received: "  + message.payload)
    r.set(message.topic,message.payload)


Connected = False
broker_addr="127.0.0.1"
port=1883

client=mqttClient.Client("RedisReceiver")
client.on_connect= on_connect
client.on_message= on_message

client.connect(broker_addr,port)

client.loop_start()
while Connected != True:
    time.sleep(0.1)
client.subscribe("node/#")
```

```
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print ("exiting")
    client.disconnect()
    client.loop_stop()
```

Penjelasan dari potongan kode program diatas adalah:

1. baris ke 1 melakukan konfigurasi koneksi pada redis.
2. baris ke 3-9 melakukan overloading pada method on_connect yang terdapat dalam library paho mqtt, jika dalam membangun koneksi berhasil maka status variabel Connected menjadi True.
3. baris ke 11-13 melakukan overloading pada method on_message yang terdapat dalam library paho mqtt, jika terdapat data publish yang masuk, akan dilakukan proses set data pada redis server variabel key dari topic sedangkan value diisi dengan payload message mqtt.
4. baris ke 16-18 melakukan inisiasi variable Connected sebagai acuan apakah sudah terhubung dengna broker atau belum, broker_addr yaitu alamat ip broker, dan port adalah alamat port broker.
5. baris ke 20-24 melakukan inisiasi variable client sebagai mqtt client, setelah itu melakukan overloading dengan method on_connect dan on_message dan melakukan koneksi ke alamat ip dan port.
6. baris ke 26-36 melakukan proses looping jika sudah terkoneksi terhadap broker maka akan melakukan subscribe.

5.4 Implementasi Database

Pada implementasi *database*, diperlukan proses update dan upgrade pada sistem linux untuk mendapatkan versi aplikasi yang terbaru. Terdapat dua aplikasi *database* yang digunakan pada penelitian ini yaitu MongoDB dan Redis. MongoDB berfungsi menyimpan data *list node* yang terdaftar beserta sensornya, sedangkan Redis menyimpan data sensor yang didapatkan dari *node sensor*.

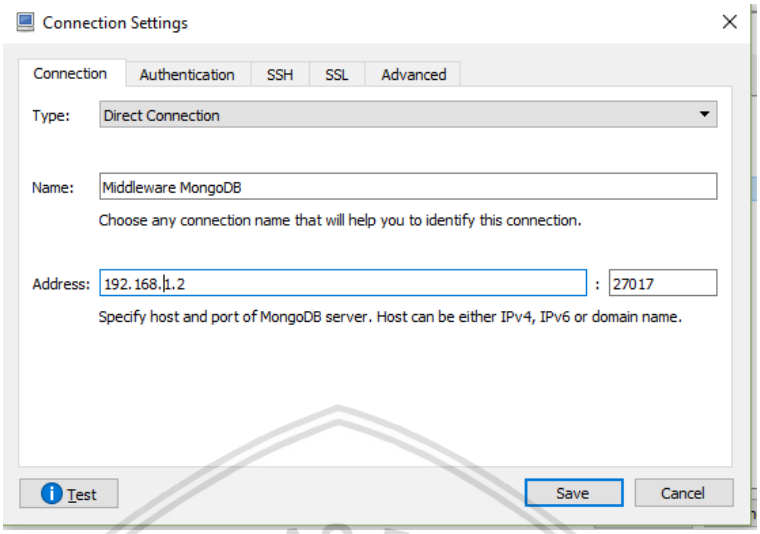
5.4.1 Implementasi MongoDB

Berikut ini adalah kode yang perlu dijalankan agar aplikasi MongoDB terpasang pada *middleware*.

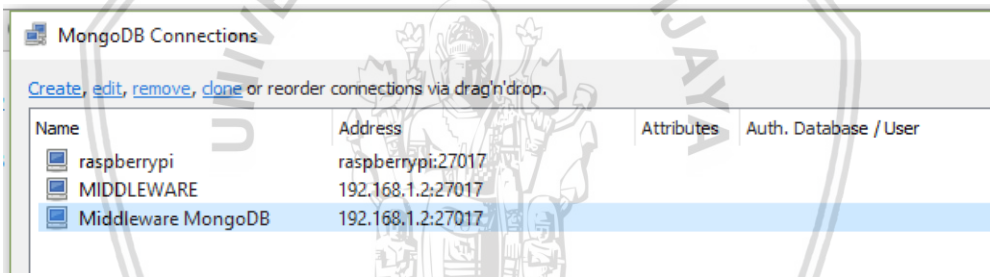
```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install MongoDB-server
```

Setelah proses instalasi selesai dilakukan, maka tahapan selanjutnya adalah membuat sebuah *database* dan sebuah *collection* yang akan digunakan dalam menyimpan data *list node sensor*. Dikarenakan MongoDB tidak menyediakan GUI dalam proses pengolahan *database* secara langsung, maka penulis memasang program tambahan seperti Robo3T untuk proses pengolahan *database*. Berikut

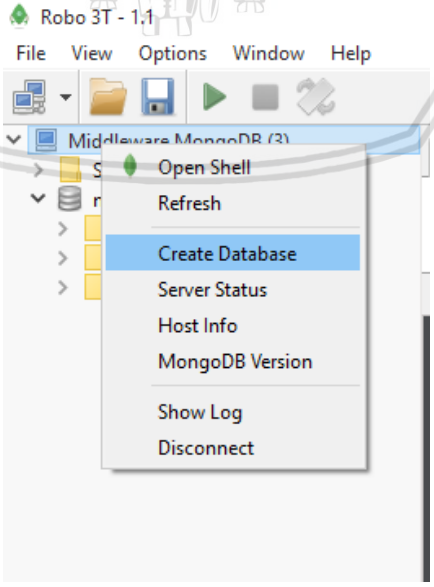
ini tahapan-tahapan yang perlu dilakukan dalam membuat database dan collection yang dibutuhkan:



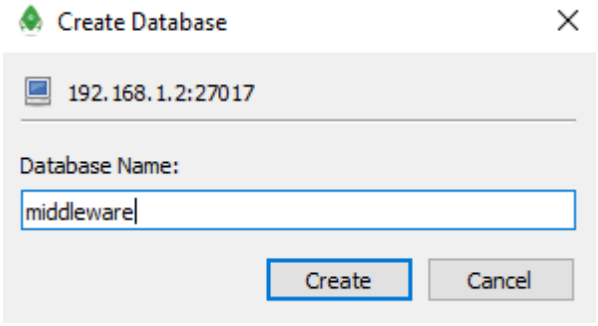
Gambar 5.1 Konfigurasi Koneksi MongoDB



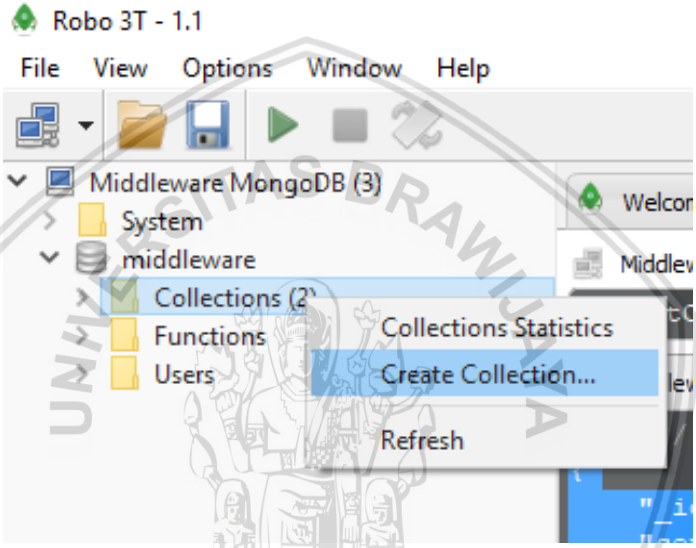
Gambar 5.2 Koneksi *Database* MongoDB



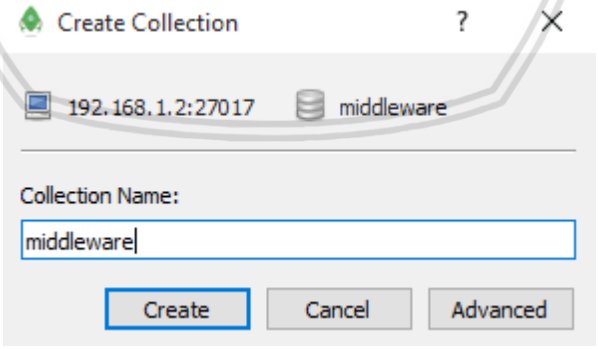
Gambar 5.3 Menu Create Database MongoDB



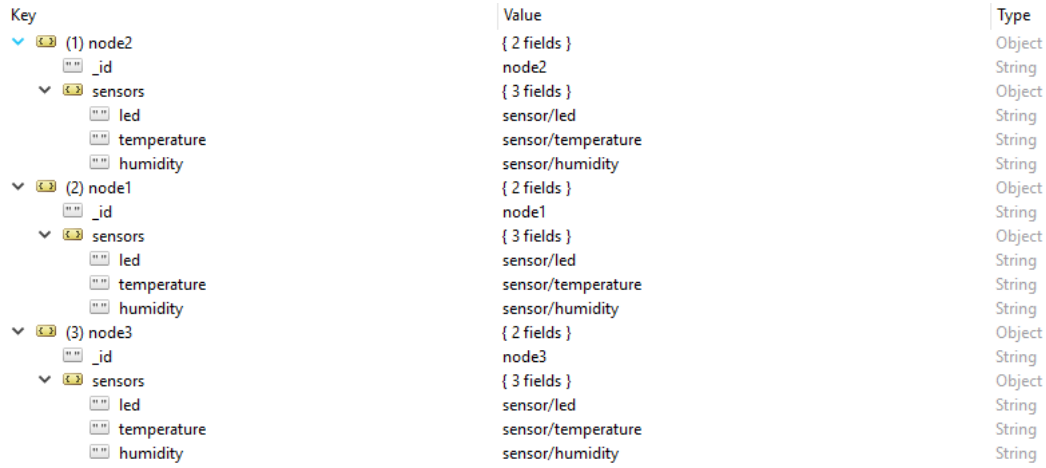
Gambar 5.4 Membuat Database middleware



Gambar 5.5 Menu Membuat Collection



Gambar 5.6 Membuat Collection middleware



Key	Value	Type
(1) node2	{ 2 fields }	Object
_id	node2	String
sensors	{ 3 fields }	Object
led	sensor/led	String
temperature	sensor/temperature	String
humidity	sensor/humidity	String
(2) node1	{ 2 fields }	Object
_id	node1	String
sensors	{ 3 fields }	Object
led	sensor/led	String
temperature	sensor/temperature	String
humidity	sensor/humidity	String
(3) node3	{ 2 fields }	Object
_id	node3	String
sensors	{ 3 fields }	Object
led	sensor/led	String
temperature	sensor/temperature	String
humidity	sensor/humidity	String

Gambar 5.7 List Data Node Sensor Setelah Proses Registrasi dari REST-Server

5.4.2 Implementasi Redis

Berikut ini adalah kode yang perlu dijalankan agar aplikasi Redis terpasang pada *middleware*.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install Redis-server
```

5.5 Implementasi Node Sensor

Pada implementasi *node sensor*, diperlukan proses pembuatan *script* yang nantinya akan dijalankan pada masing-masing node sensor. Pembuatan *script* ini berfokus kepada bagaimana menjalankan protokol MQTT sebagai arsitektur *publish subscribe* terhadap *middleware*, selain itu ditambahkan beberapa sensor yang dibutuhkan seperti led dan dht11. Pada akhirnya *node sensor* melakukan *publish data sensor* dan mendapatkan perintah dari proses *subscribe*.

5.5.1 Implementasi Node Sensor Arduino

```
#define sensor 0
dht DHT;
const char* ssid     = "REST-Middleware";
const char* password = "zxcvbnm123";
const char* mqtt_server = "192.168.1.2";
WiFiClient espClient;
PubSubClient client(espClient);
int sensortemp = 1;
int sensorhum = 1;

void callback(char* topic, byte* payload, unsigned int length) {
 Serial.print("Message arrived [");
 Serial.print(topic);
 Serial.print("] ");

 String strTopic = String((char*)topic);
 if (strTopic == "node/node2/sensor/led")
   for (int i=0;i<length;i++) {
    char receivedChar = (char)payload[i];
    Serial.print(receivedChar);
```

```
21      if (receivedChar == '0')
22
23      digitalWrite(LED_BUILTIN, HIGH);
24      if (receivedChar == '1')
25       digitalWrite(LED_BUILTIN, LOW);
26      }
27    else if(strTopic == "node/node2/sensor/temperature")
28      for (int i=0;i<length;i++) {
29      char receivedChar = (char)payload[i];
30      Serial.print(receivedChar);
31      if (receivedChar == '0')
32
33        sensortemp=0;
34      if (receivedChar == '1')
35        sensortemp=1;
36      }
37    else if(strTopic == "node/node2/sensor/humidity")
38      for (int i=0;i<length;i++) {
39      char receivedChar = (char)payload[i];
40      Serial.print(receivedChar);
41      if (receivedChar == '0')
42
43        sensorhum=0;
44      if (receivedChar == '1')
45        sensorhum=1;
46      }
47    Serial.println();
48  }
49
50  void reconnect() {
51
52   while (!client.connected()) {
53   Serial.print("Attempting MQTT connection...");
54
55   if (client.connect("WEMOS Client")) {
56    Serial.println("connected");
57
58    client.subscribe("node/node2/sensor/led");
59    client.subscribe("node/node2/sensor/temperature");
60    client.subscribe("node/node2/sensor/humidity");
61   } else {
62    Serial.print("failed, rc=");
63    Serial.print(client.state());
64    Serial.println(" try again in 5 seconds");
65
66    delay(5000);
67    }
68   }
69  }
70
71  void setup() {
72    WiFi.mode(WIFI_STA);
73    digitalWrite(LED_BUILTIN, HIGH);
74    pinMode(LED_BUILTIN, OUTPUT);
75    Serial.begin(115200);
76    delay(10);
77    // We start by connecting to a WiFi network
78    Serial.println();
79    Serial.print("Connecting to ");
80    Serial.println(ssid);
81
82    WiFi.begin(ssid, password);
83
84    while (WiFi.status() != WL_CONNECTED) {
85      delay(500);
```

```
86      Serial.print("RECONNECTING");
87      Serial.println();
88    }
89
90    Serial.println("");
91    Serial.println("WiFi connected");
92    Serial.println("IP address: ");
93    Serial.println(WiFi.localIP());
94
95    client.setServer(mqtt_server, 1883);
96    client.setCallback(callback);
97  }
98
99  void loop() {
100   if(!client.connected()){
101     reconnect();
102   }
103
104   client.loop();
105   delay(2000);
106   int ledstatus = digitalRead(LED_BUILTIN);
107   if(ledstatus==1){
108     client.publish("node/node2/sensor/led/status","0");
109
110   }else{
111     client.publish("node/node2/sensor/led/status","1");
112   }
113
114   DHT.read11(sensor);
115   char temp[50];
116   char lembab[50];
117   int tempint = int(DHT.temperature);
118   int lembabint = int(DHT.humidity);
119   String t = String(tempint);
120   String h = String(lembabint);
121   t.toCharArray(temp,50);
122   h.toCharArray(lembab,50);
123   if(sensortemp==1){
124     client.publish("node/node2/sensor/temperature/status",temp);
125
126   }else{
127     client.publish("node/node2/sensor/temperature/status","OFF");
128   }
129   if(sensorhum==1){
130     client.publish("node/node2/sensor/humidity/status",lembab);
131
132   }else{
133     client.publish("node/node2/sensor/humidity/status","OFF");
134   }
135 }
```

Berikut ini adalah penjelasan dari potongan kode diatas:

1. Baris ke 1-9 inisiasi variabel pin dht, dht, ssid wifi, password wifi, mqtt_server, wifi client, sensortemp, dan sensorhum
2. Baris ke 11-48 melakukan *overriding* method callback yang terdapat pada library mqtt ketika ada sebuah publish data yang diterima oleh arduino maka akan melakukan perintah terhadap sensor, seperti menghidupkan dan mematikan sensor led, temperatur dan kelembaban.

3. Baris ke 50-69 inisiasi *method reconnect*, apabila arduino tidak terhubung dengan *broker* maka akan melakukan proses *reconnect* dan melakukan *subscribe* ke beberapa *topic*.
4. Baris ke 71-97 inisiasi *method setup* yaitu *method* yang dijalankan pertama kali pada saat Arduino dihidupkan, proses yang dilakukan adalah menghubungkan ke jaringan wifi lalu membangun koneksi ke *broker* MQTT dan melakukan *setcallback* jika terdapat *publish* data yang masuk.
5. Baris ke 99-135 inisiasi *method loop* yaitu *method* yang dijalankan secara berkala, ketika tersambung dengan *broker* mqtt maka arduino akan melakukan *publish* data secara berkala dengan *topic* yang sesuai dengan nama *sensor* masing-masing dan *payload* data yang berisi nilai dari *sensor* tersebut.

5.5.2 Implementasi Node Sensor NodeMCU

Dalam implementasi script NodeMCU dibagi menjadi dua yaitu init.lua dan mqtt.lua. File init.lua adalah script yang pertama kali dijalankan ketika NodeMCU dihidupkan, sedangkan mqtt.lua adalah script yang melakukan proses publish dan subscribe data pada broker MQTT. Berikut adalah contoh script yang diimplementasikan pada NodeMCU:

init.lua

```
1  station_cfg={}
2  station_cfg.ssid="REST-Middleware"
3  station_cfg.pwd="zxcvbnm123"
4
5  wifi.setmode(wifi.STATION)
6  wifi.sta.config(station_cfg)
7  wifiStatusPrev = wifi.STA_IDLE
8  print("WiFi: Connecting to "..station_cfg.ssid)
9  wifi.sta.connect()
10
11 --wifi reconnect logic timer
12 tmr.alarm(0, 500, 1, function()
13     wifiStatus = wifi.sta.status()
14     if wifiStatus ~= wifiStatusPrev then
15         print("WiFi:      status      change      "..wifiStatusPrev.."-
16 >"..wifiStatus)
17     end
18     if (wifiStatusPrev ~= 5 and wifiStatus == 5) then
19         print("WiFi: connected")
20         print(wifi.sta.getip())
21     elseif (wifiStatus ~= 1 and wifiStatus ~= 5) then
22       print("WiFi: Reconnecting to "..WIFI_SSID)
23       wifi.sta.connect()
24     end
25     wifiStatusPrev = wifiStatus
26 end)
27 dofile("mqtt.lua")
```

Berikut ini adalah penjelasan dari potongan kode diatas:

1. Baris 1-9 melakukan proses membangun koneksi dengan *access point*.
2. Baris 11-26 melakukan inisiasi *timer reconnect wifi*, apabila koneksi *wifi* terputus maka akan membangun koneksi ulang dengan *access point*.

3. Baris 27 melakukan eksekusi pada *file* mqtt.lua.

	mqtt.lua
1	MQTT_BrokerIP = "192.168.1.2"
2	MQTT_BrokerPort = 1883
3	MQTT_ClientID = "NODEMCU-1"
4	
5	led2 = 4
6	gpio.write(led2, gpio.HIGH)
7	gpio.mode(led2, gpio.OUTPUT)
8	d=require('dht11')
9	d.init(5)
10	
11	--initialize sensor status = on
12	sensortemp = 1
13	sensorhum = 1
14	
15	m = mqtt.Client(MQTT_ClientID, 120)
16	
17	m:on("message", function(client, topic, data)
18	print("MQTT: "..topic .. ":")
19	if topic=="node/node1/sensor/led" then
20	if data ~= nil then
21	if data=="1" then
22	gpio.write(led2, gpio.LOW)
23	elseif data=="0" then
24	gpio.write(led2, gpio.HIGH)
25	end
26	end
27	elseif topic=="node/node1/sensor/temperature" then
28	if data ~= nil then
29	print(data)
30	if data=="1" then
31	sensortemp=1
32	elseif data=="0" then
33	sensortemp=0
34	end
35	end
36	elseif topic=="node/node1/sensor/humidity" then
37	if data ~= nil then
38	print(data)
39	if data=="1" then
40	sensorhum=1
41	print(data)
42	sensorhum=0
43	end
44	end
45	end
46	end)
47	
48	m:on("offline", function
49	(client)
50	publishMqtt:stop()
51	print("MQTT: offline")
52	end)
53	
54	--MQTT reconnect logic timer
55	reconnMqtt = tmr.create()
56	
57	reconnMqtt:register(1, tmr.ALARM_SEMI, function (t)
58	reconnMqtt:interval(500);
59	print("MQTT: trying to connect to
60	"..MQTT_BrokerIP..":"..MQTT_BrokerPort);
61	m:close()
62	m:connect(MQTT_BrokerIP, MQTT_BrokerPort, 0, 1, function(client)

```
63      print("MQTT: connected")
64      publishMqtt:start()
65      print("PUBLISH: STARTED")
66
67      print("SUBSRIBE : STARTED")
68      m:subscribe("node/node1/sensor/led",0,                  function(conn)
69  print("subscribe LED success") end)
70      m:subscribe("node/node1/sensor/temperature",0,          function(conn)
71  print("subscribe TEMP success") end)
72      m:subscribe("node/node1/sensor/humidity",0,             function(conn)
73  print("subscribe HUM success") end)
74   end, function(client, reason)
75      --publishMqtt:stop()
76      print("MQTT: connection failed with reason "..reason)
77      reconnMqtt:start()
78   end)
79  end)
80
81  --MQTT local timestamp publishing timer
82  publishMqtt = tmr.create()
83  publishMqtt:register(2, tmr.ALARM_AUTO, function (t)
84    publishMqtt:interval(5000);
85
86    ledstatus = 0
87    tempstatus = "ON"
88    humstatus = "ON"
89
90    if gpio.read(led2) == 0 then
91      ledstatus=1
92    else
93      ledstatus=0
94    end
95
96    if sensortemp == 0 then
97      tempstatus="OFF"
98    else
99      tempstatus=d.getTemp()
100   end
101
102   if sensorhum == 0 then
103     humstatus="OFF"
104   else
105     humstatus=d.getHumidity()
106   end
107
108   m:publish("node/node1/sensor/led/status",    ledstatus,    0,    0,
109 function(client) print("DATA DIKIRIMKAN : ", ledstatus) end)
110   m:publish("node/node1/sensor/temperature/status",  tempstatus,  0,
111 0, function(client) print("DATA DIKIRIMKAN : ", tempstatus) end)
112   m:publish("node/node1/sensor/humidity/status",  humstatus,  0,  0,
113 function(client) print("DATA DIKIRIMKAN : ", humstatus) end)
114   print("PUBLISH DATA")
115 end)
116
117 reconnMqtt:start()
```

Berikut ini adalah penjelasan dari potongan kode diatas:

1. Baris 1-13 inisiasi koneksi ke *broker*, serta inisiasi pin gpio yang digunakan untuk mendapatkan data *sensor*.
2. Baris ke 15 melakukan koneksi mqtt ke *broker* dengan parameter yang sebelumnya telah diinisiasi.

3. Baris ke 17-46 inisiasi method on message dari *library* mqtt yang berfungsi apabila menerima sebuah *publish* data maka akan melakukan perintah terhadap sensor, seperti menghidupkan dan mematikan sensor led, temperatur dan kelembaban.
4. Baris ke 48-52 inisiasi *method* on offline dari *library* mqtt yang berfungsi menghentikan publish ketika offline.
5. Baris ke 55-79 membuat method timer yaitu *method* yang akan menjalankan fungsinya secara berkala, fungsi dari *method* reconnMqtt ini adalah melakukan proses *reconnect* apabila status koneksi mqtt tidak terhubung serta melakukan *subscribe* dan memanggil fungsi *publish* data.
6. Baris ke 82-115 membuat *method timer* publishMqtt, fungsi dari *method* ini adalah melakukan publish data sensor secara berkala ke *broker*.
7. Baris ke 117 melakukan eksekusi *timer* pada reconMqtt.

5.5.3 Implementasi Node Sensor RaspberryPI

```
1   GPIO.setwarnings(False)
2   GPIO.setmode(GPIO.BCM)
3   GPIO.cleanup()
4   LED_PIN = 18
5   GPIO.setup(LED_PIN, GPIO.OUT)
6
7   global sensortemp
8   global sensorhum
9   global sensorled
10
11  sensortemp = 1
12  sensorhum = 1
13  sensorled = 0
14
15  Broker = "192.168.1.2"
16  sub_topic1 = "node/node3/sensor/led"
17  sub_topic2 = "node/node3/sensor/temperature"
18  sub_topic3 = "node/node3/sensor/humidity"
19
20  sensordht = dht11.DHT11(pin=4)
21
22  def on_connect(client, userdata, flags, rc):
23      print("Connected with result code "+str(rc))
24      client.subscribe(sub_topic1)
25      client.subscribe(sub_topic2)
26      client.subscribe(sub_topic3)
27
28  def on_message(client, userdata, msg):
29      message = str(msg.payload)
30      print(msg.topic+" "+message)
31      if msg.topic == "node/node3/sensor/led":
32          if msg.payload == "1":
33              global sensorled
34              sensorled = 1
35              GPIO.output(LED_PIN, True)
36          elif msg.payload == "0":
37              global sensorled
38              sensorled = 0
39              GPIO.output(LED_PIN, False)
```

```
40      elif msg.topic == "node/node3/sensor/temperature":
41          if msg.payload == "1":
42              global sensortemp
43              sensortemp = 1
44              #sensor temp on
45          elif msg.payload == "0":
46              global sensortemp
47              sensortemp = 0
48              #sensor temp off
49      elif msg.topic == "node/node3/sensor/humidity":
50          if msg.payload == "1":
51              global sensorhum
52              sensorhum = 1
53              #sensor humidity on
54          elif msg.payload == "0":
55              global sensorhum
56              sensorhum = 0
57              #sensor humidity off
58
59
60  def on_publish(mosq, obj, mid):
61      print("DATA :  " + str(mid))
62
63
64  client = mqtt.Client()
65  client.on_connect = on_connect
66  client.on_message = on_message
67  client.connect(Broker, 1883, 60)
68  client.loop_start()
69
70  while True:
71      hasil = sensordht.read()
72      if sensortemp=="0":
73          client.publish("node/node3/sensor/temperature/status",
74  "SENSOR OFF")
75          print("Temp: " + "SENSOR OFF")
76      else:
77          if hasil.is_valid():
78              client.publish("node/node3/sensor/temperature/status",
79  hasil.temperature)
80              print("Temp: " + str(hasil.temperature))
81
82      if sensorhum=="0":
83          client.publish("node/node3/sensor/humidity/status",   "SENSOR
84  OFF")
85          print("Hum: " + "SENSOR OFF")
86      else:
87          if hasil.is_valid():
88              client.publish("node/node3/sensor/humidity/status",
89  hasil.humidity)
90              print("Hum: " + str(hasil.humidity))
91
92      if sensorled=="0":
93          client.publish("node/node3/sensor/led/status", "SENSOR OFF")
94          print("Led: " + "SENSOR OFF")
95      else:
96          client.publish("node/node3/sensor/led/status", sensorled)
97          print("Led: " + str(sensorled))
98      time.sleep(5)
```

Berikut ini adalah penjelasan dari potongan kode diatas:

1. Baris ke 1-20 inisiasi pin GPIO, variabel sensor, alamat broker, topik mqtt, dan sensor DHT.

2. Baris ke 22-26 inisiasi *method* on_connect yaitu *method* yang dijalankan ketika terhubung dengan mqtt *broker*, dalam *method* ini menjalankan proses *subscribe* pada beberapa topik mqtt.
3. Baris 28-57 inisiasi *method* on_message yaitu *method* yang apabila menerima sebuah *publish* data maka akan melakukan perintah terhadap sensor, seperti menghidupkan dan mematikan sensor led, temperatur dan kelembaban.
4. Baris 60-61 inisiasi *method* on_publish yaitu *method* yang dijalankan apabila telah sukses melakukan *publish* data.
5. Baris 64-68 inisiasi koneksi mqtt terhadap *broker* dan melakukan *overriding method* on_message dan on_publish yang terdapat pada *library* mqtt.
6. Baris 70-98 melakukan sebuah *looping* yang berfungsi *mempublish* data sensor secara berkala ke *broker*.

BAB 6 PENGUJIAN DAN HASIL ANALISIS PENGUJIAN

Pada bab ini dijelaskan bagaimana proses pengujian dan analisis hasil pengujian dari REST *middleware* publish-*subscribe* yang telah dibuat. Pengujian dilaksanakan untuk memberi jawaban apakah REST *middleware publish-subscribe* yang telah dikembangkan dapat menyelesaikan masalah yang terdapat pada pendahuluan maupun pada rumusan masalah.

6.1 Perancangan Pengujian

Perancangan pengujian membahas tentang bagaimana scenario dan parameter yang digunakan dalam proses pengujian. Pada penelititan ini pengujian dibagi menjadi dua yaitu pengujian fungsional dan pengujian performa.

6.1.1 Perancangan Pengujian Fungsional

Pada perancangan pengujian fungsional memfokuskan kepada apakah sistem sudah berjalan baik dan benar. Tingkat keberhasilan pengujian ini ditentukan oleh keluaran yang didapat apakah sesuai dengan yang diharapkan. Berikut adalah tabel skenario pengujian fungsional yang dilakukan pada penelitian ini:

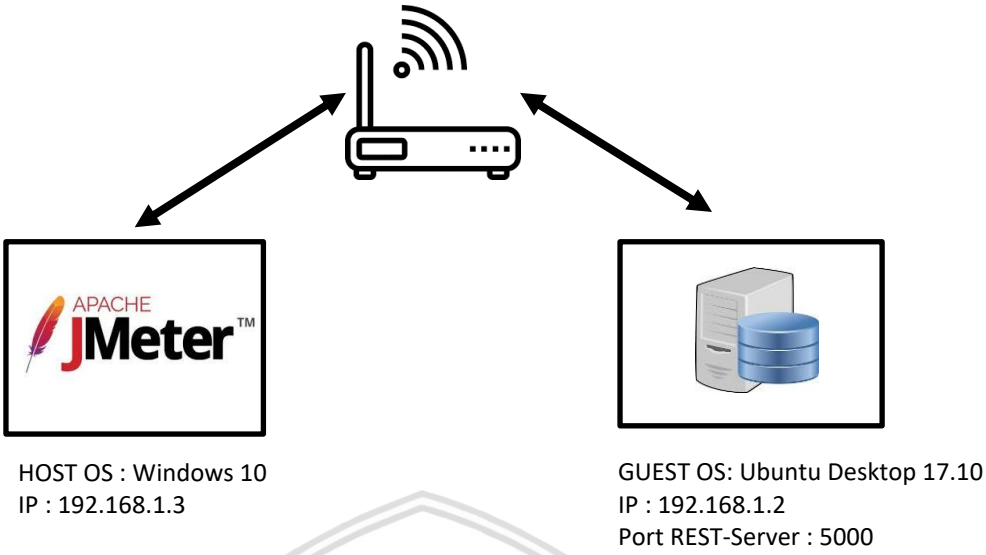
Tabel 6.1 Skenario Pengujian Fungsional

No	Deskripsi Pengujian	Skenario
PF_001	Pengujian mengambil daftar *node* yang terdaftar beserta *sensor*nya	1. User mengirimkan *request* HTTP GET . 2. *Middleware* mengembalikan status HTTP 200 beserta *node* dan *sensor*.
PF_002	Pengujian mengambil daftar *sensor* yang terdaftar pada *node* yang dipilih	1. User mengirimkan *request* HTTP GET. 2. *Middleware* mengembalikan status HTTP 200 beserta daftar *sensor*.
PF_003	Pengujian mengambil daftar *sensor* beserta data live *sensor* pada *node* yang dipilih	1. User mengirimkan *request* HTTP GET. 2. *Middleware* mengembalikan status HTTP 200 beserta daftar *sensor* dan data live *sensor*.
PF_004	Pengujian mengambil data live *sensor* pada satu *sensor* yang dipilih	1. User mengirimkan *request* HTTP GET. 2. *Middleware* mengembalikan status HTTP 200 beserta data live *sensor*.

PF_005	Pengujian registrasi *node* beserta *sensor*nya	1 User mengirimkan *request* HTTP POST dengan payload JSON: *node*, label dan *topic*. 2. *Middleware* mengembalikan status HTTP 200 beserta pesan "Success Adding Node".
PF_006	Pengujian penambahan *sensor* dari *node* yang sudah terdaftar	1. User mengirimkan *request* HTTP POST dengan payload JSON: *node*, label dan *topic*. 2. *Middleware* mengembalikan status HTTP 200 beserta pesan "Success Adding Sensor".
PF_007	Pengujian mengirim perintah pada *node* dan *sensor* yang dipilih	1. User mengirimkan *request* HTTP POST dengan payload JSON: <*sensor*> sesuai dengan nama *sensor*nya. Contoh *sensor* temperatur maka key JSONnya menjadi "temperatur" pula 2. *Middleware* mengembalikan status HTTP 200 beserta pesan "Sukses mengirim perintah ke *node sensor*".

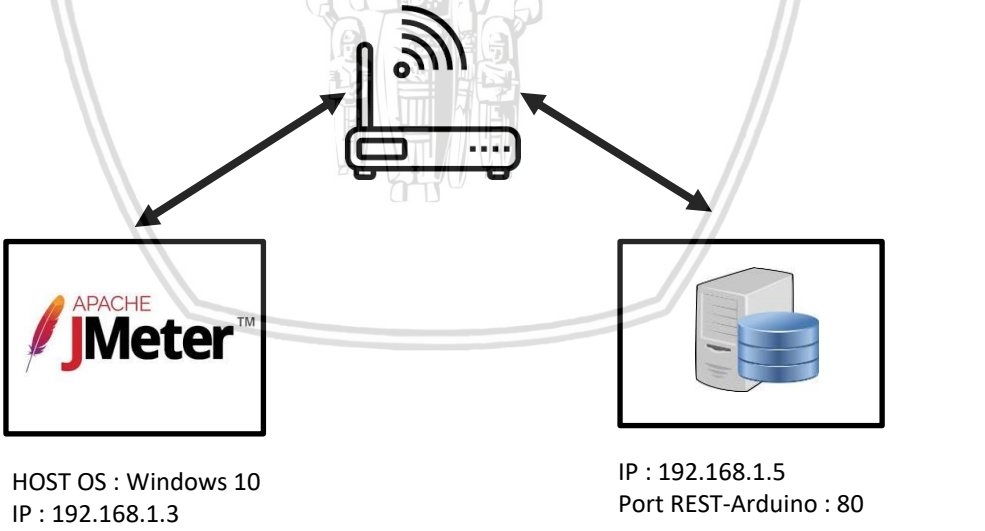
6.1.2 Perancangan Pengujian Performa

Setelah melakukan pengujian fungsionalitas, peneliti menguji performa dari sistem yang telah dibuat. Pengujian dilakukan dengan mengamati komunikasi antar entitas menggunakan beberapa parameter yang telah ditentukan antara lain *latency, error rate*. Pengujian performa ini dibagi menjadi tiga bagian, pertama penguji melakukan HTTP *request get* dan *post* pada REST server *middleware*, kedua melakukan HTTP *request get* dan *post* pada REST server Arduino dan yang ketiga yaitu melakukan pengujian *overhead* antara kedua protokol. Adapun scenario yang diterapkan pada pengujian pertama dan kedua adalah dengan menggunakan jumlah HTTP *request* yang dikirimkan secara bersamaan dengan jumlah *request* 50, 100 dan 150 user. Aplikasi yang digunakan dalam pengujian performa ini adalah Apache JMeter. Sedangkan pada pengujian yang terakhir Aplikasi yang digunakan dalam mengukur besaran overhead messagenya adalah Wireshark.



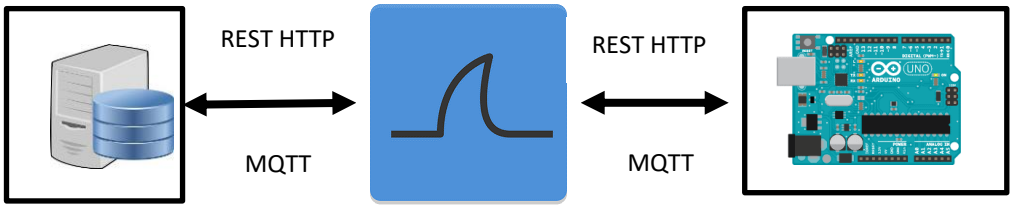
Gambar 6.1 Rancangan Pengujian Performa REST Middleware

Pada Gambar 6.1 Aplikasi JMeter berjalan pada HOST OS sebagai penguji kami menggunakan sebuah laptop dengan sistem operasi Windows 10 dan berada pada alamat IP 192.168.1.3 sedangkan pada sisi Midleware REST-server berjalan pada *port* 5000 dan menggunakan alamat *ip static* dengan alamat 192.168.1.2.



Gambar 6.2 Rancangan Pengujian Performa REST Node Sensor

Pada Gambar 6.2 Aplikasi JMeter berjalan pada HOST OS sebagai penguji kami menggunakan sebuah laptop dengan sistem operasi Windows 10 dan berada pada alamat IP 192.168.1.3 sedangkan pada sisi *node sensor* REST-server berjalan pada *port* 80 dan mendapatkan *ip dhcp* dengan alamat 192.168.1.5.



MIDDLEWARE
IP : 192.168.1.2
Port REST-Server : 5000
MQTT Broker Port : 1883

Arduino / Wemos D1
IP : 192.168.1.4

Gambar 6.3 Rancangan Pengujian Perbandingan Overhead HTTP dan MQTT

Pada Gambar 6.3 membandingan overhead protokol REST HTTP dan MQTT pada proses melakukan koneksi dalam menerima perintah, setelah itu membandingkan hasilnya.

Tabel 6.2 Skenario Pengujian Performa

No	Parameter	Keterangan	Hasil yang diharapkan
PP_001	*Latency* yang terjadi saat proses GET data oleh user pada REST *server* pada *middleware*	Menggunakan *tools* JMeter untuk melakukan proses mendapatkan data dengan variasi jumlah *user* sebesar 50, 100, dan 150 *user*	Average Latency < 1000ms
PP_002	*Latency* yang terjadi saat proses GET data oleh user pada REST *server* pada *Arduino*	Menggunakan *tools* JMeter untuk melakukan proses mendapatkan data dengan variasi jumlah *user* sebesar 50, 100, dan 150 *user*	Average Latency < 1000ms
PP_003	*Latency* yang terjadi saat proses POST data oleh user hingga diterima REST *server* pada *Middleware*	Menggunakan *tools* JMeter untuk melakukan pengiriman data dengan variasi jumlah *user* sebesar 50, 100, dan 150 *user*	Average Latency < 1000ms
PP_004	*Latency* yang terjadi saat proses POST data oleh user hingga diterima REST *server* pada *Arduino*	Menggunakan *tools* JMeter untuk melakukan pengiriman data dengan variasi jumlah *user* sebesar 50, 100, dan 150 *user*	Average Latency < 1000ms
PP_005	Jumlah *user* yang	Menggunakan tools JMeter untuk	Average

	dapat ditangani oleh REST-Server pada Middleware	melakukan pengiriman data dengan variasi jumlah *user* sebesar 50, 100, dan 150 user	Error Rate < 20%
PP_004	Jumlah user yang dapat ditangani oleh REST-Server pada Arduino	Menggunakan tools JMeter untuk melakukan pengiriman data dengan variasi jumlah *user* sebesar 50, 100, dan 150 user	Average Error Rate < 20%

6.2 Hasil Pengujian dan Analisis

Berikut ini hasil dari proses pengujian baik pengujian fungsional dan pengujian performa:

6.2.1 Hasil Pengujian dan Analisis Pengujian Fungsional

Dalam hasil pengujian fungsional ditentukan oleh dua nilai yaitu berhasil dan tidak berhasil. Jika pengujian terhadap suatu fitur sukses tanpa error maka pengujian fungsional tersebut berhasil. Berikut ini adalah tabel hasil pengujian fungsional:

Tabel 6.3 Hasil Pengujian Fungsional

No	Deskripsi Pengujian	Hasil
PF_001	Pengujian mengambil daftar *node* yang terdaftar beserta *sensor*nya	Berhasil
PF_002	Pengujian mengambil daftar *sensor* yang terdaftar pada *node* yang dipilih	Berhasil
PF_003	Pengujian mengambil daftar *sensor* beserta data live *sensor* pada *node* yang dipilih	Berhasil
PF_004	Pengujian mengambil data live *sensor* pada satu *sensor* yang dipilih	Berhasil
PF_005	Pengujian registrasi *node* beserta *sensor*nya	Berhasil
PF_006	Pengujian penambahan *sensor* dari *node* yang sudah terdaftar	Berhasil
PF_007	Pengujian mengirim perintah pada *node* dan	Berhasil

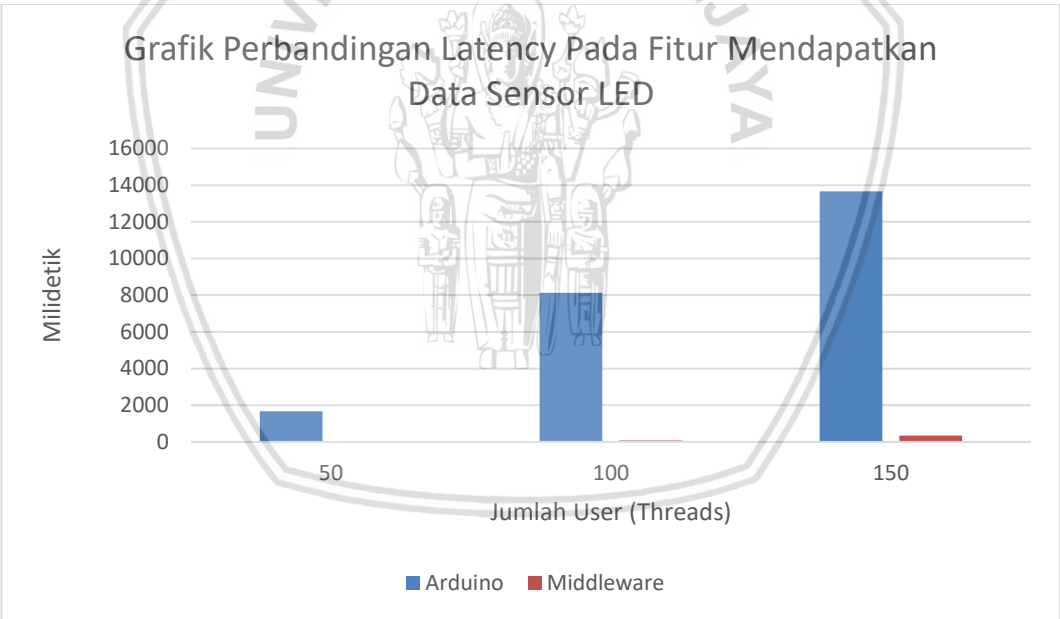
	sensor yang dipilih	

6.2.2 Hasil Pengujian dan Analisis Pengujian Performa

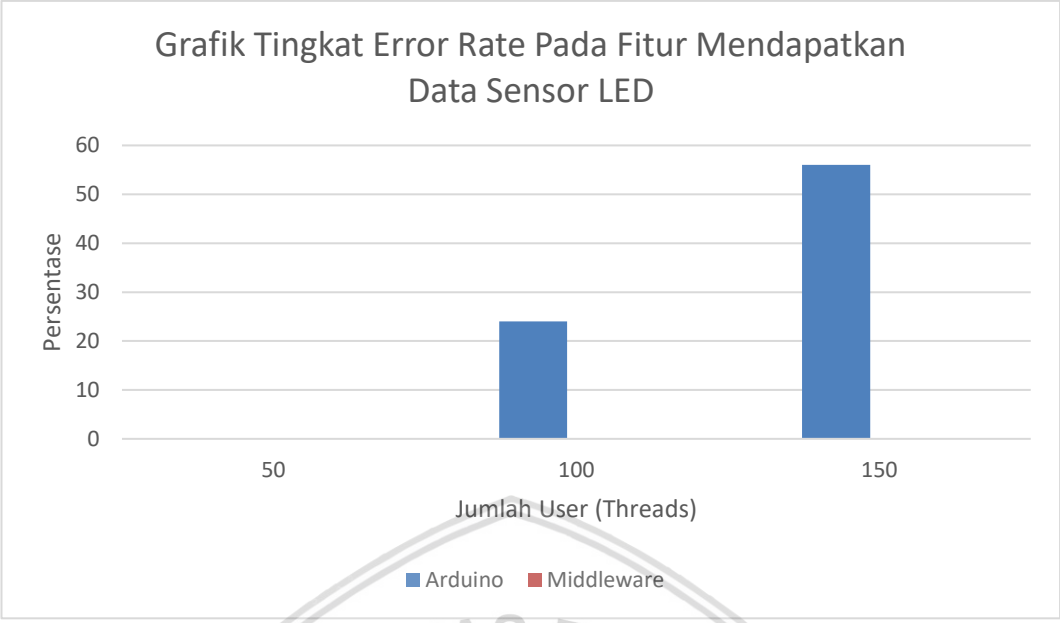
Pengujian performa yang pertama yaitu pengujian latency dan error rate yang terjadi saat melakukan proses GET data pada REST-Server yang terdapat pada middleware dan REST-Server yang terdapat pada Arduino. Pengujian dilakukan dengan menggunakan aplikasi J-Meter dengan variasi user sebanyak 50,100 dan 150 user. Hasil pengujian dapat dilihat pada Tabel berikut:

Tabel 6.4 Hasil Pengujian Average Latency dan Error Rate pada GET Data

User	Average Latency (ms) pada REST-Arduino	Average Latency (ms) pada REST-Middleware	Error Rate (%) pada REST-Arduino	Error Rate (%) pada REST-Middleware
50	1666	4	0	0
100	8132	89	24	0
150	13661	350	56	0



Gambar 6.4 Grafik Perbandingan Latency pada GET Data

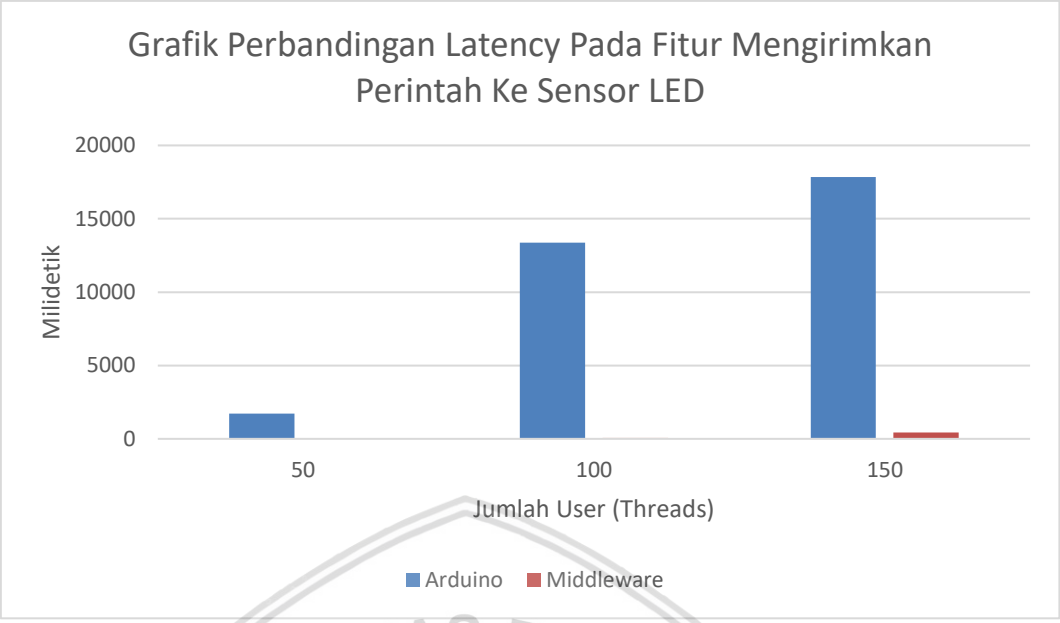


Gambar 6.5 Grafik Perbandingan Error Rate pada GET Data

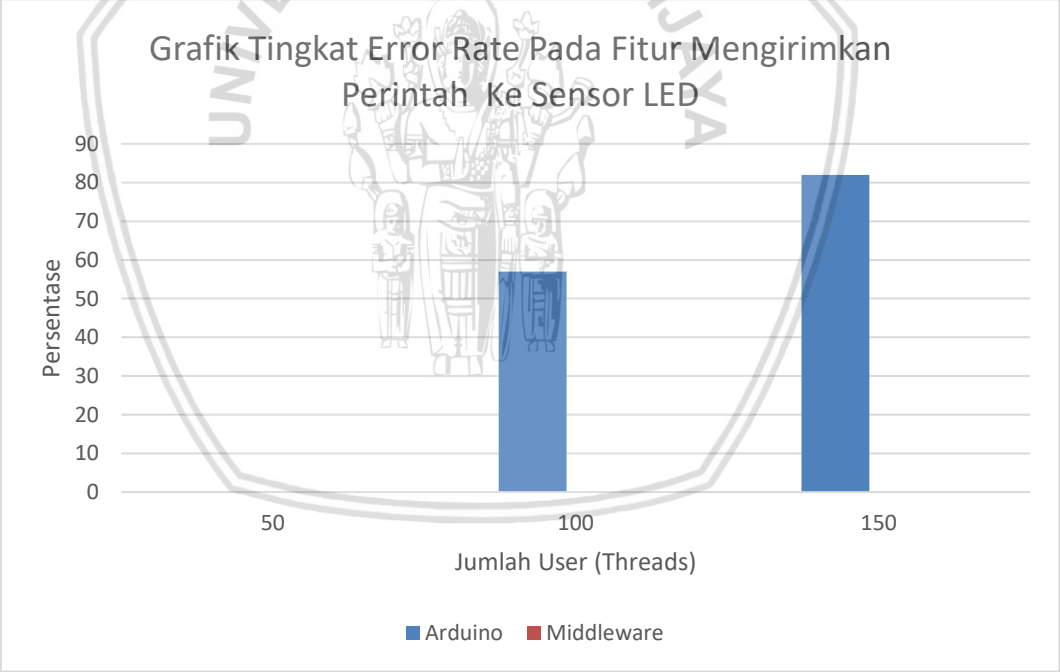
Pengujian performa yang kedua yaitu pengujian perbandingan latency dan error rate yang terjadi saat melakukan proses POST data pada REST-Server yang terdapat pada middleware dan REST-Server yang terdapat pada Arduino. Pengujian dilakukan dengan menggunakan aplikasi J-Meter dengan variasi user sebanyak 50,100 dan 150 user. Hasil pengujian dapat dilihat pada Tabel berikut:

Tabel 6.5 Hasil Pengujian Average Latency dan Error Rate pada POST Data

User	Average Latency (ms) pada REST-Arduino	Average Latency (ms) pada REST-Middleware	Error Rate (%) pada REST-Arduino	Error Rate (%) pada REST-Middleware
50	1727	8	0	0
100	13375	52	57	0
150	17841	441	82	0



Gambar 6.6 Grafik Perbandingan Latency pada POST Data

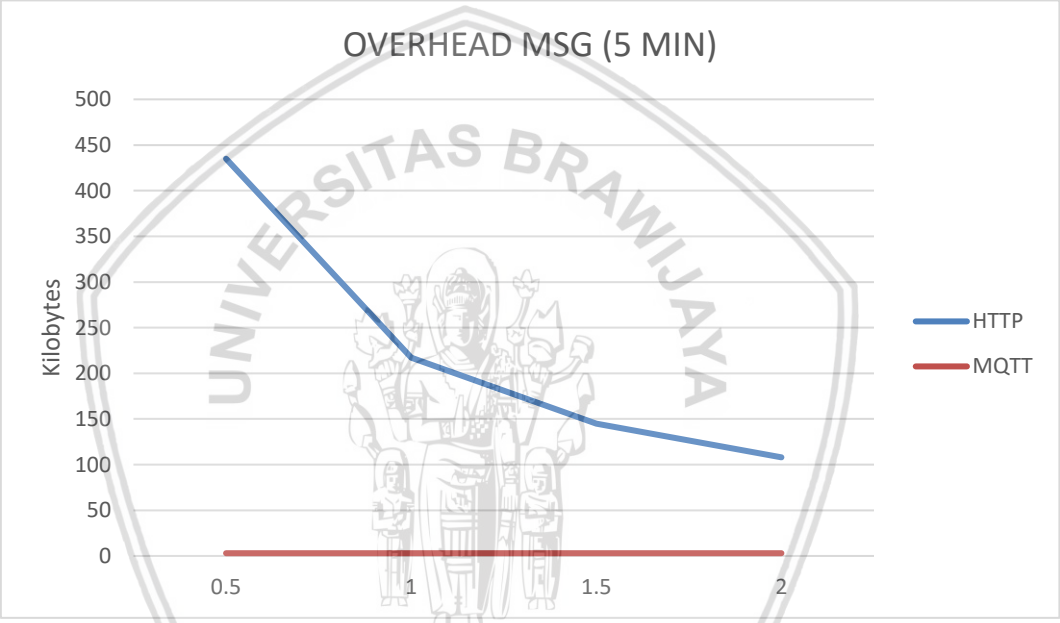


Gambar 6.7 Grafik Perbandingan Error Rate pada POST Data

Pengujian performa yang ketiga yaitu pengujian perbandingan overhead pada kedua protokol pada proses melakukan koneksi dalam menerima perintah. Pengujian dilakukan dengan menggunakan aplikasi Wireshark dengan variasi waktu request 0.5, 1, 1.5 dan 2 detik. Hasil pengujian dapat dilihat pada Tabel berikut:

Tabel 6.6 Hasil Pengujian Overhead

Periode Request (Detik)	Besaran Overhead dalam waktu 5 menit (KB)	
	HTTP	MQTT
0.5	450	3
1	217	3
1.5	145	3
2	108	3



Gambar 6.8 Grafik Perbandingan Overhead Message pada HTTP dan MQTT

BAB 7 KESIMPULAN DAN SARAN

Pada bab ini menjelaskan tentang kesimpulan yang didapatkan peneliti dari proses penelitian yang telah dilakukan sebelumnya.

7.1 Kesimpulan

Berdasarkan hasil dari tahap-tahapan perancangan, implementasi dan pengujian yang telah dilakukan oleh peneliti, maka berikut ini kesimpulan yang dapat diambil:

1. Alur data dari pengguna sampai ke perangkat node sensor, yaitu ketika pengguna mengirimkan sebuah HTTP POST *request* ke *middleware* yang berisi sebuah perintah. Setelah REST server menerima sebuah request dari pengguna maka REST server akan melakukan proses *publish* ke *broker*. Setelah itu *broker* akan meneruskan perintah ke *subscriber* atau *node sensor.*
2. Alur data dari node sensor hingga sampai ke pengguna memiliki beberapa tahap yaitu, pertama *node sensor* akan melakukan sebuah *publish* data sensor. Setelah itu *subscriber middleware* akan menyimpan data sensor tersebut kedalam *database* redis. Jika pengguna mengirimkan sebuah HTTP GET *request* data sensor ke *middleware,* maka REST server melakukan pengambilan data sensor pada *database* redis. Setelah itu REST server mengirimkan sebuah HTTP response kepada pengguna beserta data sensor.
3. Mekanisme translasi perintah yang dilakukan *middleware* pada awalnya user mengirimkan perintah berupa HTTP POST dengan payload JSON yang nantinya oleh REST-*server* diubah menjadi sebuah *publish* data terhadap *broker* dan oleh *broker* diteruskan kepada node sensor.
4. Dari pengujian yang telah dilakukan pada bab sebelumnya pada pengujian fungsional telah berfungsi sesuai dengan apa yang dijelaskan dan dirancang pada bab analisis kebutuhan dan perancangan. Sedangkan pada pengujian performa dapat disimpulkan sebagai berikut:
    a. Berdasarkan pengujian *latency* menggunakan J-Meter dengan scenario 50, 100, 150 *request* dibuktikan dengan hasil ujicoba penggunaan *middleware* menghasilkan performa rata-rata *latency* dibawah 1000ms dan tanpa *middleware* menghasilkan performa rata-rata *latency* diatas 1000ms.
    b. Berdasarkan pengujian *error rate* menggunakan J-Meter dengan scenario 50, 100, 150 *request* dibuktikan dengan hasil ujicoba penggunaan *middleware* menghasilkan performa rata-rata *error rate* dibawah 1% dan tanpa *middleware* menghasilkan performa rata-rata *error rate* hingga 82%.
    c. Dengan adanya *middleware* sebagai penengah antara user dengan perangkat *IoT*, membuat kinerja dari perangkat *IoT* lebih ringan, karena

jika jumlah user yang mengakses meningkat maka tidak berdampak pada perangkat *IoT*.

7.2 Saran

Berdasarkan kesimpulan penelitian yang telah dijabarkan sebelumnya, maka penulis memiliki beberapa saran untuk penelitian selanjutnya yang berhubungan dengan topik penelitian ini. Berikut ini adalah saran penulis:

1. Penelitian berikutnya dapat mengembangkan sistem REST-*Middleware* dengan menggunakan protokol IoT yang berbeda, terutama dalam hal komunikasi antara *middleware* dengan *node sensor*.
2. REST-*Middleware* yang telah dibangun dapat dikembangkan lebih lanjut dalam hal keamanan pada protokol HTTP dan MQTT.
3. Pengujian performa dapat dilakukan dengan menguji aspek lainnya salah satu contohnya seperti *security*.

DAFTAR PUSTAKA

Al-Fuqaha, A. et al., 2015. Internet of Things: A Survey on Enabling Technologies, Protocols and Application. *IEEE Communications Surveys & Tutorials,* 17(4), pp. 2347 - 2376.

Amaran, M. H., Noh, N. A. M., Rohmad, M. S. & Hashim, H., 2015. A Comparison of Lightweight Communication Protocols in Robotic. *Procedia Computer Science 76,* pp. 400-405.

Andre, 2017. *Pengertian Bahasa Pemrograman C.* Diakses 10 September 2017, dari http://www.duniailkom.com/tutorial-belajar-c-pengertian-bahasa-pemrograman-c/

Aziz, B., 2015. A formal model and analysis of an IoT protocol. *Ad Hoc Networks.*

Belajarpython, 2017. *Belajarpython - Tutorial Python Bahasa Indonesia.* Diakses 10 September 2017, dari https://www.belajarpython.com/

Chodorow, K. & Dirolf, M., 2010. *MongoDB : The Definitive Guide.* Sebastopool: O'Reilly Media.

Farahzadi, A., Shams, P., Rezazadeh, J. & Farahbakhsh, R., 2017. Middleware technologies for cloud of things-a survey. *Digital Communications and Networks* .

Flask, 2017. *Foreword - Flask Documentation.* Diakses 17 Desember 2017, dari http://flask.pocoo.org/docs/0.12/foreword/#what-does-micro-mean

Gao, L., Zang, C. & Sun, L., 2011. RESTful Web of Things API in Sharing Sensor Data. *Internet Technology and Applications (iTAP).*

Grinberg, M., 2014. *Flask Web Development.* California: O'reilly Media.

Guinard, D. & Trifa, V., 2009. Towards the Web of Things: Web Mashups for Embedded Devices. In: *Towards the Web of Things: Web Mashups for Embedded Devices.* s.l.:s.n., pp. 1506-1518.

Hunkeler, U., Truong, H. L. & Clark, A. S., 2015. A Publish/Subscribe Protocol For Wireless Sensor Network. *IEEE.*

Khare, R. & Taylor, R. N., 2004. Extending the REpresentational State Transfer (REST) Architectural Style for Decentralized Systems. *International Conference on Software Engineering.*

Lua, 2017. *Lua: about.* Diakses 10 September 2017, dari: https://www.lua.org/about.html

Macedo, T. & Oliveira, F., 2011. *Redis Cookbook.* California: O'Reilly Media.

Mainetti, L., Mighali, V. & Partono, L., 2015. A Software Architecture Enabling the Web of Things. *IEEE INTERNET OF THINGS JOURNAL,* 2(6), pp. 445-454.

MQTT, 2014. *Documentation | MQTT.* Diakses 27 Desember 2017, dari http://mqtt.org/documentation

Nelson, J., 2016. *Mastering Redis.* Birmingham: Packt Publishing.

Nurohman, P., 2016. *Kenapa Kamu Harus Memilih Mempelajari Bahasa Pemograman Python - CodePolitan.com.* Diakses 10 September 2017, dari https://www.codepolitan.com/kenapa-kamu-harus-memilih-bahasa-pemograman-python-57cdd334db9c2-18512

Paganelli, F., Turchi, S. & Giuli, D., 2016. A Web of Things Framework for RESTful Applications and Its Experimentation in a Smart City. *IEEE SYSTEMS JOURNAL,* 10(4), pp. 1412-1423.

Patel, K. K. & Patel, S. M., 2016. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. *International Journal of Engineering Science and Computing,* 6(5), pp. 6122-6131.

Redis, 2017. *Redis.* Diakses 17 Desember 2017, dari https://redis.io/

Sandoval, J., 2009. *RESTful Java Web Services.* Birmingham: Packt Publishing.

Xiao, G., Guo, J., Xu, L. D. & Gong, Z., 2014. User Interoperability With Heterogeneous IoT Devices Through Transformation. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* 10(2), pp. 1486-1496.

Zeng, D., Guo, S. & Cheng, Z., 2011. The Web of Things: A Survey. *JOURNAL OF COMMUNICATIONS,* 6(6), pp. 424-438.